NetLogger Manual

COLLABORATORS			
	TITLE : NetLogger Manual		*REFERENCE* :
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Dan Gunter	October 3, 2008	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Preface 9
1.1 Conventions . . . 9

2 Overview 9
2.1 Methodology: Logging Best Practices . . . 9
2.2 Tools . . . 10

3 Installation 10
3.1 System requirements . . . 10
3.2 Install C . . . 11
3.3 Install Java . . . 11
3.4 Install PERL . . . 11
3.5 Install Python . . . 11
3.6 Install R . . . 12

4 Instrumentation APIs 12
4.1 C API . . . 12
4.2 Java API . . . 13
4.3 Perl API . . . 13
4.4 Python API . . . 13

5 NetLogger Pipeline 13
5.1 Syslog-NG . . . 13
5.2 Log Parser (nl_parser) . . . 15
5.2.1 Parser modules . . . 15
5.2.2 Configuration . . . 17
5.2.2.1 Global section . . . 17
5.2.2.2 Parser module sections . . . 18
5.2.3 Running . . . 19
5.2.3.1 Standalone mode . . . 19
5.2.3.2 Pipeline mode . . . 19
5.3 Log Loader (nl_loader) . . . 19
5.3.1 Schema . . . 19
5.3.1.1 Example . . . 21
5.3.2 Configuration . . . 21
5.3.2.1 global section . . . 21
5.3.2.2 input section . . . 21
5.3.2.3 database section . . . 22

5.3.3 Running . . . 23
5.3.3.1 Standalone mode . . . 23
5.3.3.2 Pipeline mode . . . 24
5.4 Pipeline management (nl_pipeline) . . . 24
5.4.1 Configuration files . . . 24
5.4.1.1 Include syntax . . . 25
5.4.1.2 Logging extensions . . . 26
5.4.1.3 Complete example . . . 27
5.4.2 Rotating files . . . 28
5.4.3 Sending messages to nl_loader and nl_parser . . . 29
5.5 SQL and R Analysis . . . 29
5.5.1 Performing SQL queries from R . . . 29
5.5.2 Creating plots with R . . . 30

6 Frequently Asked Questions **31**

A Tool manual pages **32**
A.1 netlogd(1) . . . 32
A.1.1 NAME . . . 32
A.1.2 SYNOPSIS . . . 32
A.1.3 DESCRIPTION . . . 32
A.1.4 OPTIONS . . . 32
A.1.5 EXAMPLES . . . 32
A.1.6 EXIT STATUS . . . 33
A.1.7 BUGS . . . 33
A.1.8 AUTHOR . . . 33
A.2 nl_check(1) . . . 33
A.2.1 NAME . . . 33
A.2.2 SYNOPSIS . . . 33
A.2.3 DESCRIPTION . . . 33
A.2.4 OPTIONS . . . 33
A.2.5 EXAMPLES . . . 33
A.2.6 EXIT STATUS . . . 34
A.2.7 BUGS . . . 34
A.2.8 RESOURCES . . . 34
A.2.9 AUTHOR . . . 34
A.3 nl_check_pipeline(1) . . . 34
A.3.1 NAME . . . 34
A.3.2 SYNOPSIS . . . 34

A.3.3 DESCRIPTION . . . 34
A.3.4 OPTIONS . . . 34
A.3.5 EXAMPLES . . . 35
A.3.6 EXIT STATUS . . . 35
A.3.7 BUGS . . . 35
A.3.8 AUTHOR . . . 35
A.4 nl_cpuprobe(1) . . . 35
A.4.1 NAME . . . 35
A.4.2 SYNOPSIS . . . 35
A.4.3 DESCRIPTION . . . 35
A.4.4 OPTIONS . . . 36
A.4.5 EXAMPLES . . . 36
A.4.6 EXIT STATUS . . . 36
A.4.7 BUGS . . . 36
A.4.8 AUTHOR . . . 36
A.5 nl_date(1) . . . 36
A.5.1 NAME . . . 36
A.5.2 SYNOPSIS . . . 36
A.5.3 DESCRIPTION . . . 36
A.5.4 OPTIONS . . . 37
A.5.5 EXAMPLES . . . 37
A.5.6 EXIT STATUS . . . 37
A.5.7 BUGS . . . 37
A.5.8 AUTHOR . . . 37
A.6 nl_dbquery(1) . . . 37
A.6.1 NAME . . . 37
A.6.2 SYNOPSIS . . . 37
A.6.3 DESCRIPTION . . . 38
A.6.4 OPTIONS . . . 38
A.6.4.1 Time Range: . . . 38
A.6.5 USAGE . . . 38
A.6.6 EXAMPLES . . . 39
A.6.7 EXIT STATUS . . . 39
A.6.8 BUGS . . . 39
A.6.9 AUTHOR . . . 39
A.7 nl_findbottleneck(1) . . . 40
A.7.1 NAME . . . 40
A.7.2 Synopsis . . . 40
A.7.3 DESCRIPTION . . . 40

A.7.4 OPTIONS . . . 40
A.7.5 EXAMPLES . . . 40
A.7.6 EXIT STATUS . . . 40
A.7.7 BUGS . . . 40
A.7.8 AUTHOR . . . 41
A.8 nl_findmissing(1) . . . 41
A.8.1 NAME . . . 41
A.8.2 SYNOPSIS . . . 41
A.8.3 DESCRIPTION . . . 41
A.8.4 OPTIONS . . . 41
A.8.5 EXAMPLES . . . 42
A.8.6 EXIT STATUS . . . 42
A.8.7 BUGS . . . 42
A.8.8 AUTHOR . . . 42
A.9 nl_ganglia(1) . . . 42
A.9.1 NAME . . . 42
A.9.2 SYNOPSIS . . . 42
A.9.3 DESCRIPTION . . . 43
A.9.4 OPTIONS . . . 43
A.9.5 EXAMPLES . . . 43
A.9.6 EXIT STATUS . . . 43
A.9.7 BUGS . . . 43
A.9.8 RESOURCES . . . 43
A.9.9 AUTHOR . . . 43
A.10 nl_interval(1) . . . 44
A.10.1 NAME . . . 44
A.10.2 SYNOPSIS . . . 44
A.10.3 DESCRIPTION . . . 44
A.10.4 OPTIONS . . . 44
A.10.5 EXAMPLES . . . 44
A.10.6 EXIT STATUS . . . 45
A.10.7 BUGS . . . 45
A.10.8 AUTHOR . . . 45
A.11 nl_loader(1) . . . 45
A.11.1 NAME . . . 45
A.11.2 SYNOPSIS . . . 45
A.11.3 DESCRIPTION . . . 45
A.11.4 OPTIONS . . . 46
A.11.4.1 Pipeline-mode options: . . . 46

A.11.4.2 Standalone-mode options: . . . 46
A.11.5 USAGE . . . 46
A.11.6 EXAMPLES . . . 47
A.11.7 EXIT STATUS . . . 47
A.11.8 BUGS . . . 47
A.11.9 RESOURCES . . . 47
A.11.10AUTHOR . . . 47
A.12 nl_notify(1) . . . 47
A.12.1 NAME . . . 47
A.12.2 SYNOPSIS . . . 47
A.12.3 DESCRIPTION . . . 48
A.12.4 OPTIONS . . . 48
A.12.5 EXAMPLES . . . 48
A.12.6 EXIT STATUS . . . 48
A.12.7 BUGS . . . 49
A.12.8 AUTHOR . . . 49
A.13 nl_parser(1) . . . 49
A.13.1 NAME . . . 49
A.13.2 SYNOPSIS . . . 49
A.13.3 DESCRIPTION . . . 49
A.13.4 OPTIONS . . . 49
A.13.4.1 General options: . . . 49
A.13.4.2 Command-line parser configuration: . . . 50
A.13.5 USAGE . . . 50
A.13.6 SIGNALS . . . 50
A.13.7 EXAMPLES . . . 50
A.13.8 EXIT STATUS . . . 51
A.13.9 BUGS . . . 51
A.13.10RESOURCES . . . 51
A.13.11AUTHOR . . . 51
A.14 nl_pipeline(1) . . . 51
A.14.1 NAME . . . 51
A.14.2 SYNOPSIS . . . 51
A.14.3 DESCRIPTION . . . 51
A.14.4 OPTIONS . . . 51
A.14.5 EXAMPLES . . . 52
A.14.6 EXIT STATUS . . . 52
A.14.7 BUGS . . . 52
A.14.8 AUTHOR . . . 52

A.15 nl_view(1) . . . 52
A.15.1 NAME . . . 52
A.15.2 SYNOPSIS . . . 52
A.15.3 DESCRIPTION . . . 53
A.15.4 OPTIONS . . . 53
A.15.5 EXAMPLES . . . 54
A.15.6 EXIT STATUS . . . 54
A.15.7 BUGS . . . 54
A.15.8 RESOURCES . . . 54
A.15.9 AUTHOR . . . 54
A.16 nl_wflowgen(1) . . . 54
A.16.1 NAME . . . 54
A.16.2 SYNOPSIS . . . 54
A.16.3 DESCRIPTION . . . 54
A.16.4 OPTIONS . . . 55
A.16.5 EXAMPLES . . . 55
A.16.6 EXIT STATUS . . . 55
A.16.7 BUGS . . . 55
A.16.8 RESOURCES . . . 55
A.16.9 AUTHOR . . . 55
A.17 nl_write(1) . . . 55
A.17.1 NAME . . . 55
A.17.2 SYNOPSIS . . . 56
A.17.3 DESCRIPTION . . . 56
A.17.4 OPTIONS . . . 56
A.17.5 EXAMPLES . . . 56
A.17.6 EXIT STATUS . . . 56
A.17.7 BUGS . . . 56
A.17.8 AUTHOR . . . 56

B Index **57**

List of Figures

1 1: NetLogger Pipeline 13
2 2: NetLogger Database Schema 20
3 3: nl_pipeline includes 25
4 4 Sample xyplot() output 31

1 Preface

This is the manual for the NetLogger Toolkit. For more details, downloads, etc. please refer to the NetLogger web pages at http://acs.lbl.gov/NetLoggerWiki/

1.1 Conventions

Italic Used for file and directory names, email addresses, and new terms where they are defined.

`Constant Width` Used for code listings and for keywords, variables, functions, command options, parameters, class names, and HTML tags where they appear in the text. In code listings, user input to the terminal will be prefixed with a "`$`".

`Constant Width Italic` Used as a general placeholder to indicate items that should be replaced by actual values.

Link text Used for URLs and cross-references.

2 Overview

Anyone who has ever tried to debug or do performance analysis of complex distributed applications knows that it can be a very difficult task. Problems may be in many various software components, hardware components, networks, the OS, etc.

NetLogger is designed to make this easier. NetLogger is both a methodology for analyzing distributed systems, and a set of tools to help implement the methodology.

2.1 Methodology: Logging Best Practices

The NetLogger methodology, also called the Logging *Best Practices* (BP), is documented in detail at http://www.cedps.net/index.php/Log
The following is a brief summary:

Terminology For clarity here are some definitions of terms which are used throughout the NetLogger documentation.

- ***event*** A uniquely named point of interest within a given system occurring at a specific time. An *event* is also a required attribute of each NetLogger log entry.
- ***log*** A file containing logging events or a stream of such events.
- ***log entry*** A single line within a log corresponding to a single event.
- ***attribute*** A detailed characteristic of an event.
- ***name/value pair*** How attributes are identified within a log entry - a *name* with the given *value* separated by an = sign.

Practices All logs should contain a unique *event* attribute and an ISO-formatted timestamp (See ISO8601). System operations that might fail or experience performance variations should be wrapped with `start` and `end` events. All logs from a given execution context should have a globally unique ID (or GUID) attribute, such as a Universal Unique Identifier (UUID) (see RFC4122). When multiple contexts are present, each one should use its own identifying attribute name ending in `.id`.

Errors A reserved `status` integer attribute must be used for all `end` events, with "0" for success and any other value for failure or partial failure. The default severity of a log message is informational, other severities are indicated with a `level` attribute.

Format Each log entry should be composed of a single line of ASCII name=value pairs (aka attributes); this format is highly portable, human-readable, and works well with line-oriented tools.

Naming For event attribute names we recommend using a '.' as a separator and go from general to specific; similar to Java class names.

A sample job submit start/end log in this format would look like the following:

```
ts=2006-12-08T18:39:19.372375Z event=org.job.submit.start user=dang job.id=37900
ts=2006-12-08T18:39:23.114369Z event=org.job.submit.end user=dang job.id=37900 status=0
```

The addition of log file grammar such as the name-value attribute pair structure encourages more regular and normalized representations than natural language sentences commonly found in ad-hoc logs.

For example, a message like `error: read from socket on foobar.org:1234: remote host baz.org:4321 returned -1` would be:

```
ts=2006-12-08T18:48:27.598448Z event=org.my.myapp.socket.read.end level=ERROR status=-1 ↩
    host=foobar.org:1234 peer=baz.org:4321
```

The open source NetLogger Toolkit is a set of tools to implement this methodology.

2.2 Tools

The tools included with NetLogger can be grouped in four main areas:

- Logging APIs: C, Java, Perl, Python, and UNIX shell
- NetLogger Pipeline: Parse, load, and analyze logs using a relational database and the R data analysis language.
- Bottleneck detection: Test disk/network for bottleneck in WAN transfers.
- Utilities: Monitoring probes, a log receiver (netlogd), and some other pieces that are occasionally useful.

3 Installation

Installation of NetLogger is broken down by language, with Python being the primary language for the tools and in particular the NetLogger Pipeline.

3.1 System requirements

Operating System NetLogger has been tested on UNIX and Mac OSX. The Python code should work on Windows with some modifications, but this is not a priority for our development.

Software

- Java 1.5 or above (http://java.sun.com) for the Java instrumentation
- Python 2.5 or above (http://www.python.org) for the Python instrumentation API and the NetLogger Pipeline
- PERL version 5 or higher (http://www.perl.org), for the PERL instrumentation API
- MySQL, PostgreSQL or SQLite for the NetLogger Pipeline, if you want to use one of those back-ends. You will also need the associated Python interface library:

  - *MySQLdb* for MySQL
  - psycopg2 or pgdb for PostgreSQL
  - sqlite3 for SQLite
- All monitored hosts should use NTP (http://www.ntp.org), or the equivalent, for clock synchronization
- For the R analysis, you need to R version 2.6.0 or higher

3.2 Install C

This installs the C instrumentation API and the nlioperf program.

```
# Unpack sources
tar xzf netlogger-c-VERSION.tar.gz
# Run configure; make; make install
cd netlogger-c-VERSION
./configure --prefix=/your_install_path
make
make install
```

3.3 Install Java

This installs the Java instrumentation API.

```
# Copy NetLogger jarfile into desired spot
cp netlogger-VERSION.jar /your_install_path/netlogger.jar
# Then set your classpath
csh% setenv CLASSPATH $CLASSPATH:/your_install_path
# .. OR ..
sh$ export CLASSPATH=$CLASSPATH:/your_install_path
```

3.4 Install PERL

This installs the PERL instrumentation API.

```
# Unpack sources
tar xzf netlogger-perl-VERSION.tar.gz
cd netlogger-perl-VERSION
# Run PERL's standard install sequence
perl Makefile.PL
make
make install
```

3.5 Install Python

This installs the Python instrumentation API, the tools for the NetLogger Pipeline, and the miscellaneous Python tools (all of which start with *nl_* and have UNIX manual pages).

- Install from source

```
# Unpack sources
tar xzvf netlogger-python-VERSION.tar.gz
cd netlogger-python-VERSION
# Run Python's standard install sequence
python setup.py build
python setup.py install
```

- Install using *easy_install*

```
easy_install netlogger-4.1.0
```

3.6 Install R

There is no NetLogger R instrumentation API, but we do use R to analyze the data (see the SQL and R analysis section). Below are some instructions for installing the R language and the packages used by NetLogger. Full documentation is available on the R web site.

1. Use a package manager
   - R packages may be available for yum and dpkg
2. Install from source
   - Install the R base package
     - Go to the R website at http://www.r-project.org/
     - Click on CRAN under Download
     - Choose a mirror
     - Pick the correct download link and follow instructions
3. Add some libraries
   - Start R
   - Choose a mirror (you only need to do this once):

```
> chooseCRANmirror()
```

   - The following packages are recommended:
     - lattice: trellis graphics library
     - RMySQL: MySQL interface
     - Hmisc: useful functions
   - Download and install the packages:

```
install.packages(c("lattice","Hmisc","RMySQL", "RSQLite"), dependencies = TRUE)
```

   - The PostgreSQL interface uses a different installation procedure:

```
source("http://bioconductor.org/biocLite.R")
biocLite("RdbiPgSQL")
# ..still working on this one!
```

4 Instrumentation APIs

NetLogger has instrumentation APIs to produce Best Practices (BP) formatted logs for C/C++, Java, Perl, and Python.

4.1 C API

The C API documentation is auto-generated from the source code using Doxygen.

It is available online from http://acs.lbl.gov/NetLogger-releases/doc/api/c-4.1.0/

4.2 Java API

The Java API documentation is auto-generated from the source code using Javadoc.

It is available online from http://acs.lbl.gov/NetLogger-releases/doc/api/java-4.1.0/

4.3 Perl API

The Perl API documentation is auto-generated from the source code using pod2html.

It is available online from http://acs.lbl.gov/NetLogger-releases/doc/api/perl-4.1.0/

4.4 Python API

The Python API documentation is auto-generated from the source code using epydoc.

It is available online from http://acs.lbl.gov/NetLogger-releases/doc/api/python-4.1.0/

5 NetLogger Pipeline

The purpose of the NetLogger Pipeline is to normalize and structure logs for easier analysis and correlation. There are four stages to the NetLogger Pipeline, as shown in the figure below.

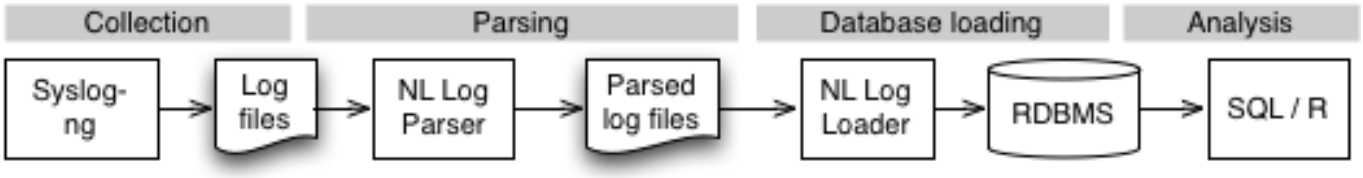

Figure 1: 1: NetLogger Pipeline

1. Log collection: we use the open source tool syslog-ng.
2. Log parsing: our tool, nl_parser, extracts information from logs and normalizes it
3. Database loading: our tool, nl_loader, loads the log entries into a relational database with a schema that is application-independent
4. Analysis: we primarily use SQL and the R data analysis language, but we are also working on web front-ends using Django.

There is also a tool that manages the nl_parser and nl_loader, which is called simply nl_pipeline.

The NetLogger Pipeline is designed so that components for each stage can also function independently of the others.

5.1 Syslog-NG

Syslog-NG (http://www.balabit.com/network-security/syslog-ng/) is a flexible and scalable system logging application that can act a as a drop-in replacement for standard syslog.

A syslog-ng server can send local data over the network (TCP or UDP), receive network data and log it locally, or do both. syslog-ng receivers can be configured to aggregate and filter logs based on program name, log level and even a regular expression on message contents. It is very scalable: if a particular receiver gets over-loaded, one can just bring up another receiver on a

another machine and send half the logs to each. syslog-ng supports fully qualified host names and time zones, which standard syslog does not. Standard syslog could also be used, but only for single site deployments.

We recommend syslog-ng 2.0 over syslog-ng 1.6 because of the new ISO date option, which is needed for logging across multiple time zones. To download, go to: http://www.balabit.com/downloads/files/syslog-ng/sources/stable/src/

Here is a commented sample syslog-ng 2.0 *sender* configuration file. For more sample configuration files, see http://www.cedps.net/index ng.

```
# Global options

options {
   # Polling interval, in ms (helps reduce CPU)
   time_sleep(50);

   # Use fully qualified domain names
   use_fqdn(yes);

   # Use ISO8601 timestamps
   ts_format(iso);

   # Number of line to buffer before writing to disk
   # (a) for normal load
   flush_lines (10);
   log_fifo_size(100);
   # (b) for heavy load
   #flush_lines (1000);
   #log_fifo_size(1000);

   # Number of seconds between syslog-ng internal stats events.
   # These are useful for watching the load.
   stats_freq(3600);
};

# Data sources: file, TCP or UDP socket, or internal

# Tail /var/log/gridftp.log, prefix copy of input with
# the prefix 'gridfp_log '
source gridftp_log {
  file ("/var/log/gridftp.log" follow-freq(1) flags(no-parse) log_prefix('gridftp_log ') );
};
# ..etc..
# Syslog-ng's own logs; for testing syslog-ng config
source syslog_ng { internal(); };

# Data sinks: file, TCP, or UDP socket

# Send "grid" logs to a remote host on TCP port 5141
destination gridlog_dst {
       tcp("remote.loghost.org" port(5141));
};
# Send other logs to a local file
destination syslog_ng_dst {
  file ("/tmp/syslog-ng.log" perm(0644) );
};

# Data pipelines
# Combine a source and a destination to make a pipeline

# Send the gridftp logs to the remote "grid" host
log {
  source(gridftp_log); destination(gridlog_dst); flags(flow-control);
};
```

```
# (and so on for the other "grid" sources)

# Send the internal logs to the local file
log {
  source(syslog_ng); destination(syslog_ng_dst);
};
```

5.2 Log Parser (nl_parser)

The nl_parser is a framework for normalizing the information in existing logs and converting them to BP format. This framework, written in Python, uses plug-in parser modules or Python-wrapped C to transform the raw logs. The parsed logs are then fed to the database loader.

See also the nl_parser manpage.

5.2.1 Parser modules

A parser module is written for each log type to be parsed. Each module must implement a class, `Parser`, with a method, `process(line)`, that takes a single line of input data and returns either an error or zero or more Python dictionaries representing the keyword/value pairs to produce. Note that the mapping between input and output is many-to-many, allowing blocks of input to be combined for one output or vice-versa. For more details on implementing your own plug-in modules, see the NetLogger Cookbook included with this dist or on-line in the documentation area under NetLogger home page - http://acs.lbl.gov/NetLoggerWiki.

A description of the parser modules distributed with NetLogger, and their parameters, follows. This same information is also available from the nl_parser program using the `-D/--desc` option.

pbs Parse contents of PBS accounting file.

Parameters:

- site {**org.mydomain**}: Site name, for site-specific processing. Current recognized sites are: *.nersc.gov = NERSC.
- suppress_hosts {True,**False**}: Do not include the list of hosts in the output. This list could be very long if the job has a high degree of parallelism.

dynamic A meta-parser that matches parser modules to a given line based on a header. The expected header is given by regular expression. For each input line, values of matching named groups, e.g. `(?P<name>expr)`, are used to select the parser to use for that line.

Parameters:

- pattern: Regular expression to extract the header
- show_header_groups {True,**False**}: A list of named groups in the header expression include in the output event. If None, False, or empty, no named header parts will not be included. If True, include any/all header parts.
- header_groups_prefix {***syslog.***}: String prefix to add to each name in the header group, to avoid name-clashes with the names already in the event record. The default prefix reflects the primary use-case of parsing a syslog-ng receiver's output.

sge_rpt Parse output file from Sun Grid Engine *reporting* logs

The parameters control which types of SGE reporting output are parsed. With no parameters, no types are parsed.

Parameters:

- host_consumable {True,**False**}: Parse the *host_consumable* type of record.

generic Generic parser that uses a fixed event name and puts all the information in a single string-valued attribute.

Parameters:

- attribute_name {***msg***}: Output name for the attribute containing the input line.
- event_name {***event***}: Output event name

gk Simplified Globus Toolkit GT2 Gatekeeper log parser.

The oddly-formatted input is transformed into a sequence of 3 events:

- globus.gatekeeper.start
- globus.gatekeeper.auth
- globus.gatekeeper.end

sge Parse output file from Sun Grid Engine (SGE)

Parameters:

- one_event {True,**False**}: If true, generate one event per SGE output record, otherwise generate a start/end event pair.

csa_acct SGI Comprehensive System Accounting (CSA) process accounting parser.

See also http://oss.sgi.com/projects/csa/.

gridftp_auth Initialize gridftp auth parser.

Parameters:

- error_events {**True**,False}: If True, hold on to transfer-starting events until a matching transfer-end is encountered. If none is found, or a transfer-end precedes a transfer-start, report a transfer-error event instead.
- error_timeout {**24h**}: How long to wait for the transfer-end event. Valid units are "s" for seconds, "m" for minutes, "h" for hours. This is ignored if error_events is False.

bestman Parse logs from Berkeley Storage Manager (BeStMan).

Parameters:

- version {1,2}: Version 1 is anything before bestman-2.2.1.r3, Version 2 is that version and later ones.
  See also http://datagrid.lbl.gov/bestman/

gensim Parse the *job* + *out* file output by Pegasus *gensim* parser.

THIS PARSER IS EXPERIMENTAL

The only way the parsercurrently runs is with BOTH the *out* and *jobs* files of the gensim output, in that order, e.g.:

```
cat out logs | nl_parser -m gensim > parsed.out
```

condorxml Parse XML condor logs.

The logs are a series of XML fragments whose outer element is <*c*>. The information in each fragment includes an event type, time, return value, resource usage stats, and bytes sent and received.

kickstart Parse the Kickstart job wrapper output.

Parameters:

- one_event {True,**False**}: If true, generate one event per kickstart invocation record, otherwise generate a start/end event pair.
- use_c {True,**False**}: Use the **experimental** C parser instead. This requires that you compiled the parser with "python setup.py swig".
  See also: http://pegasus.isi.edu/

wsgram Globus Toolkit GT4 WS-GRAM log parser

See also http://www.globus.org/toolkit/docs/4.0/execution/wsgram/developer-index.html.

bp Parse Best-Practices logs into Best-Practices logs.

Parameters:

- has_gid {True,**False**}: If true, the "gid=" keyword in the input will be replaced by the currently correct "guid=".

- verify {**True**,False}: Verify the format of the input, otherwise simply pass it through without looking at it.

gridftp Parse GridFTP (server) transfer logs.

Parameters:

- one_event {**True**,False}: If true, produce a single event for the transfer, and if False produce a start/end event pair.

jobstate Pegasus/dagman/condor jobstate parser.

Parameters:

- add_guid {True,**False**}: Add a unique identifer, using the guid= attribute, to each line of the output. The same identifier is used for all output from one instance (i.e. one run of the nl_parser).

5.2.2 Configuration

The configuration file uses an enhanced version of the ConfigObj format (the enhancement is an "include" functionality, detailed in the nl_pipeline section). The general layout is sections with [square_brackets] containing name=value pairs.

The configuration file is broken into sections. The top-level sections are: *global*, *logging*, and one section per parser module. Only the global and parser module sections are described here; the logging section is elaborated in a separate section, below.

5.2.2.1 Global section

eof_event Flag to append a special end-of-file NetLogger event when closing or rotating the file. Usually used with the *rotate* option. Default is False.

files_root Path to prepend to *files* paths. This can be overridden inside the parser sections. Default is the current directory (“.”).

modules_root Module path to prepend to module names. Default is “netlogger.parsers.modules”.

post_path Colon-separated module path to put after the system path. Default value is “” (empty).

pre_path Colon-separated module path to put before the system path. Default value is “” (empty).

rotate Time between rotations of the file. Default, zero (0), disables rotation. Units given can be seconds, minutes, hours, or days, each of which can be spelled out or abbreviated by its first letter. If enabled, all output filenames will have ".<NUM>" added to them, where <NUM> is chosen to be the next-lowest-number in the same directory as the file. For example, if the output file name is "/tmp/foo.log" and there is already a "/tmp/foo.log.3" and "/tmp/foo.log.5" at startup, then the first output file name will be "/tmp/foo.log.6".

state_file Save state to the given file. Default value is “/tmp/netlogger_parser_state”. Use “” or “None” to disable. When there is a state file, then the following occurs:

- At startup, nl_parser attempts to read the specified state file for the current positions. If this file does not exist, a warning will be printed, and it will be created.
- When reconfiguration is triggered by a signal, the state is first saved and then restored.
- When the program exits gracefully, state is first saved. In addition, the state is saved periodically (every time all files reach EOF), so even termination with SIGKILL will, in general, not lose much information.

tail Flag indicating whether to tail the input files forever. Default is False.

throttle Proportion (0 < X <= 1) of "full speed" to which the parser should throttle itself. Default is 1. This only has an effect if the data starts or stays ahead of the parser; it should not slow processing down if the parser only has short bursts of activity to perform. The value is a number in the interval (0,1], i.e. greater than 0 and less than or equal to 1. Note that the parser is single-threaded so on a dual CPU machine it can only get at most 50% of the available CPU (etc.).

use_system_path If *no*, do not include the normal Python system path to find modules. Default is True.

5.2.2.2 Parser module sections

Each parser is in its own section. The name of the section can be anything except “global” or “DEFAULT”. There are two types of parser sections. The first one parses an entire file with the same parser logic; the second one decides on the parser to use line-by-line.

1. Parser per log file. The following keywords and subsections are recognized:

   files_root Each section optionally specifies its own value for this. Default value is *files_root* from the [global] section.

   files File pattern, or list of patterns, that selects the input files. This is concatenated to the files_root value and then matched by UNIX *glob* semantics. Note that this matching is done only during initial configuration, so new files that match the pattern will not be "seen" until the program is restarted or re-configured. Default is “” (empty).

   [[<module>]] Sub-section whose name is the name of the Python parser module to use. This module name has the modules_root value prepended.

   [[[parameters]]] Additional keyword, value pairs to be passed to the module at initialization time. The meaning of these keywords is module-specific. Default is “” (empty).

   Example:

   ```
   [foo]
   [[pbs]]
   files = pbs*.txt
   [[[param]]]
   # parameters for module
   site = pdsf.lbl.gov
   ```

2. Parser per line. Each of these sections specifies a regular expression that extracts the header of each line and assigns names to parts of that header. Then the actual parser module is selected by matching to values of the (named) parts of the header. The following keywords and subsections are recognized:

   files Same as per-file.

   pattern Regular expression used to extract the header from each line. Named pattern groups in the expression use the Python *re* module syntax of “(?P<name>PATTERN)” to extract things matching “PATTERN” as group “name”. Other regular expression syntax may be used, but these named groups are important because they are used in the subsequent [[[match]]] sub-section.

   [[<module>]] As for the per-file section, a sub-section whose name is the name of the Python parser module to use.

   [[[parameters]]] Same as per-file sub-sub-section.

   [[[match]]] Keyword, value pairs that describe which headers should be matched to this parser. If empty, match anything. The keywords should be the same as the names of the named patterns given in the PATTERN expression. The values are themselves regular expressions matched against the corresponding strings extracted from the header. The first, and only the first, module to match a given header is used to parse that header’s log line.

   Example:

   ```
   [myparser]
   files = syslog-output*.log
   pattern = " (?P<level>[A-Z]+)/(?P<app>):"
   # The following all match on the value
   # of the named group, 'app'
   [[bestman]]
   [[[match]]]
   app = "bestman"
   [[pbs]]
   [[[match]]]
   app = "PBS"
   [[generic]]
   [[[match]]]
   # everything
   ```

5.2.3 Running

The nl_parser runs in one of two modes: “standalone” and “pipeline”. The standalone mode is for batch processing. A single set of files is parsed by a single parser module, producing a single output file. In pipeline mode, the parser is configured from a file. In this mode, it can read and/or tail many sets of files, applying different parser modules to each.

5.2.3.1 Standalone mode

To run in standalone mode, provide the name of the parser module and the input/output files (if none, standard input is used). Results are written to standard output.

```
$ nl_parser -m pbs input.pbs > output.bp
```

All the command-line options in standalone mode have equivalent options in the configuration file.

```
# Configuration          Command-line option
[global]
pre_path = "."           # -x/--external
state_file = x           # -r/--restore x
throttle = x             # -t/--throttle x
[mod1]
[[xyz]]                  # -m/--module xyz
files = x,y,z            # args: x y z
pattern = "\S+: "        # -e/--expr ❶
[[[parameters]]]
name = value             # -p/--param name=value
```

❶ The header is stripped, but still only one parser is used for the file.

5.2.3.2 Pipeline mode

Running in pipeline mode is usually done for you by the nl_pipeline program. It is sometimes useful, though, to run with a configuration file and the **-n/—no-action** option, as this will report errors in the configuration file:

To run in pipeline mode, with -n:

```
$ nl_parser -n -c config_file
```

5.3 Log Loader (nl_loader)

The nl_loader loads BP formatted logs into a (generic) schema.

See also the nl_loader manpage.

5.3.1 Schema

The schema used by nl_loader is shown in Figure 2, below. The name for this class of schemas is an Entity-Attribute-Value or EAV schema.

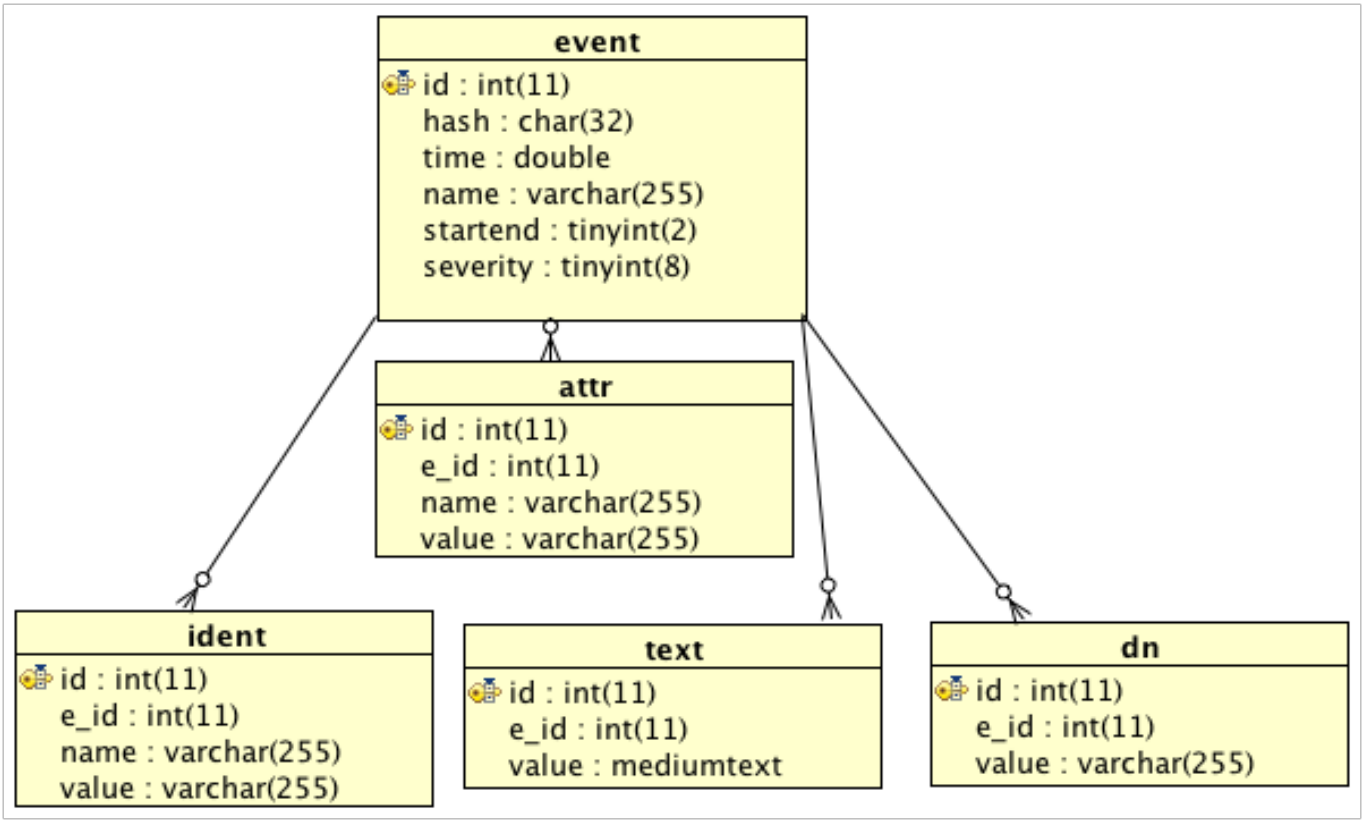

Figure 2: 2: NetLogger Database Schema

As the figure indicates, the main tables are *event* and *attr*. Below is a description of the columns in those tables.

EVENT TABLE DESCRIPTION

id Sequential identifier, used in attr

hash Hash of entire event, for uniqueness constraints

time Timestamp of event (ts field in BP logs)

name Name of event (event field in BP logs)

startend Code 0=.start event, 1=.end event, 2=neither

severity Numeric log severity: 1=FATAL, 2=ERROR, 3=WARN, 4=INFO, 5=DEBUG, 6=DEBUG+1, ..etc.. up to 255

ATTR TABLE DESCRIPTION

id Sequential identifier, auto-generated

e_id *id* value of parent event

name Attribute name

value Attribute value (up to 255 bytes)

The other tables are special attribute tables. The *ident* table is for attributes ending in ".id", such as "job.id". The nl_loader strips the .id suffix from the attribute name before inserting it in this table. As a special case, the attribute "guid" is also inserted (with name="guid") in this table.

The *text* table is for the attribute *text*, which hopefully is being used for values longer than 255 characters. Similarly, the *dn* table is for the attribute *DN*, which stands for "distinguished name".

5.3.1.1 Example

To show how a Best Practices log event gets broken up and inserted, if one were to run nl_loader with the following two events as input:

```
ts=2008-09-16T21:52:16.385281Z event=run.start level=Info job.id=123 dn=mydn user=dang guid ↩
    =BADDECAF
ts=2008-09-16T21:52:23.849174Z event=run.end level=Info status=0 msg="what a ride" guid= ↩
    BADDECAF
```

the resulting database tables would look similar to the following:

- **event** table

id	hash	time	name	startend	severity
1	259c4f1…	1221601936.38	run	0	4
2	4f9c875…	1221601943.84	run	1	4

- **attr** table

id	e_id	name	value
1	1	user	dang
2	2	status	0
3	2	msg	what a ride

- **ident** table

id	e_id	name	value
1	1	job	123
2	1	guid	BADDECAF
3	2	guid	BADDECAF

- **dn** table

id	e_id	value
1	1	mydn

5.3.2 Configuration

The configuration file uses an enhanced version of the ConfigObj format (the enhancement is an "include" functionality, detailed in the nl_pipeline section). The general layout is sections with [square_brackets] containing name=value pairs.

The configuration file is broken into sections. The top-level sections are: *global*, *input*, *database*, and *logging*. Only the first three are described here; the logging section is elaborated in a separate section, below.

5.3.2.1 global section

state_file Save state to the given file. This preserves the name and offset within the current numbered input file. The database connection (and related parameters) is not saved. Default is "/tmp/netlogger_loader_state".

5.3.2.2 input section

delete_old_files If true, delete files after loading them into the database. Overrides *move_files_dir* and *move_files_suffix*. Default is False.

filename Input filename or, for numbered files, base filename. Required.

move_files_dir If true, move files to the given directory after loading them. Overrides *move_files_suffix*. Default is False.

move_files_suffix If true, rename files by appending the given suffix after loading them. Default is False.

numbered_files Whether files have a *.##* suffix used to determine their order. This is the convention followed by the nl_loader. Default is False.

5.3.2.3 database section

uri Database connection URI. Required. The following URL schemes are recognized:

mysql://host[:port] MySQL database. Host and port could also be specified as parameters.

postgres://host[:port] PostgreSQL database. Host and port could also be specified as parameters.

sqlite:///path/to/file SQLite database file.

test:///path/to/file Dump SQL statements to file.

batch Load batch size, same as -b/—batch option. Default is 100

create If *1*, create database on load. If *2* drop and then create the database. Default is *0* (do neither).

unique Whether a *UNIQUE* constraint should be enforced on all events. This can add time to the load, but eliminates the problem of duplicate events. Default is True

schema_file Absolute or relative path to an alternative schema configuration file. This file describes the SQL statements to execute when creating new tables and when loading is finished. Default value is inside the netlogger package, under the netlogger/analysis directory in a file named *schema.conf*.

- The format of this file is the now-familiar ConfigObj format). Within this file, there is a section named for each database backend (mysql, postgres, sqlite) and inside each of these sections there are two subsections: *init* and *finalize*. Values in these subsections are SQL statements to run before any data is loaded and after all data is loaded, respectively. The *init* section has all the "CREATE TABLE" statements to create the schema. The finalize section is intended to do additional indexing, etc., on the loaded database.
- Multiple SQL values can be present in the *init* and finalize subsections. The corresponding keywords should be one or more words, in alphabetical order, separated by an underscore, e.g., "index_unique". The particular value selected for schema initialization or finalization will match the keywords provided (or used by default) to the *schema_init* and *schema_finalize* options.
- So, for example, if the schema configuration file had the following lines:

```
[mysql]
[[init]]
cookie_monster = """CREATE TABLE cookies ... """
elmo = """CREATE TABLE piano ... """
```

Then the user would (explicitly) select the first with:

```
$ nl_loader -u mysql://localhost --create --schema-init=cookie,monster ...
```

and the second with:

```
$ nl_loader -u mysql://localhost --create --schema-init=elmo ...
```

schema_init Type of schema initialization to use, encoded as a comma-separated list of keys. Which keys are available depends on the configuration file (see schema_file option). In the default configuration file, the convention is that, if keyword `X` turns on feature X, then keyword `noX` turns off feature X. The built-in features are:

index/noindex Index specified columns (MySQL and PostgreSQL)

unique/nounique Add UNIQUE constraint to event.hash column (All back-ends)

schema_finalize Type of schema finalization to use, encoded as a comma-separated list of keys. The matched statements are executed right before nl_loader exits. The intent is to allow one-time post-load actions, such as compression of the database or indexing at the end of the load. Which keys are available depends on the database engine. Default is the first keyword in the file, which in the default configuration file is "noop" for all databases.

[[parameters]] Subsection for database keyword = value parameters. Some of these may be required to successfully connect to the database:

database Name of database to which to connect. Ignored by SQLite and Test back-ends, otherwise required.

db Synonym for "database".

- For PostgreSQL, the database is required. When the user creates the schema, the database does not exist beforehand. The nl_loader works around this by connecting to the built-in "postgres" database, creating the new database, then reconnecting to the new database and continuing.

host, port, user, password Corresponding database parameter. Ignored by SQLite and Test back-ends.

- For MySQL, the *~/.my.cnf* file, if it exists, will be used for username and password information. If this file provides the *database* option and there is no such parameter, this value will be used as the database.

dsn Used instead of the part after the `://` in the connection URI.

5.3.3 Running

The program runs in two modes: "standalone" and "pipeline". In standalone mode, you provide a list of files to load, and provide the connection URI, database parameters, etc. via command-line options. In pipeline mode, you are assumed to be running the nl_parser program to create a **series** of output files that the nl_loader is then loading into the database. The use of one mode or the other is signaled by whether the `-u/--uri` option (for standalone mode) or `-c/--config` option (for pipeline mode) is given.

5.3.3.1 Standalone mode

To run in standalone mode:

```
$ nl_loader -u DB-URI ...
```

All the command-line options in standalone mode have equivalent options in the configuration file.

```
# Configuration         Command-line option
[global]
state_file = x          # -r/--restore x
[input]
filename = x            # -i/--input x
numbered_files = no     # (implied)
[database]
uri = sqlite:///x       # -u/--uri sqlite:///x
batch = 10              # -b/--insert-batch 10
create = 1              # -C/--create
create = 2              # -D/--drop
unique = no             # -U/--unique
unique = yes            # (-U not given)
schema_file = x         # -s/--schema-file x
schema_init = x,y,z     # --schema-init x,y,z
schema_finalize = x,y   # --schema-finalize x,y
[[parameters]]
name = value            # -p/--param name=value
```

5.3.3.2 Pipeline mode

Running in pipeline mode is usually done for you by the nl_pipeline program. It is sometimes useful, though, to run with a configuration file and the **-n/—no-action** option, as this will report errors in the configuration file:

To run in pipeline mode, with -n:

```
$ nl_loader -n -c config_file
```

5.4 Pipeline management (nl_pipeline)

The nl_pipeline program is intended to run the nl_loader and nl_parser together as a system service, parsing and loading logs as they arrive on disk from, for example, a syslog-ng receiver. It does very little itself besides fork off the loader and parser; most of the work is done by extra features in these two programs. However, because these features are almost exclusively for use in "pipeline" mode, they are documented in this section instead of the nl_loader or nl_parser sections. These features are: file rotation, actions on signals or UDP messages, and the include and logging extensions to the configuration file syntax.

5.4.1 Configuration files

The nl_pipeline program runs the nl_parser and nl_loader using configuration files. Rather than taking the names of the two files, it enforces a convention: the configuration files must be in the same directory and must be named "nl_parser.conf" and "nl_loader.conf". Therefore, rather than taking the name of a configuration file, the nl_pipeline program's `-c/--config` option takes the name of a directory, e.g., `nl_pipeline -c /path/to/dir/`.

There are several other files read or written by the nl_pipeline:

- PID files: nl_loader.pid and nl_parser.pid
- log files: nl_pipeline.log, and also nl_loader and nl_parser log files

If the configuration directory is specified as follows:

```
nl_pipeline -c $conf_dir/etc
```

then the default layout of files would be:

```
$conf_dir
  |
  +-- etc
  |    |
  |    +-- nl_loader.conf
  |    +-- nl_parser.conf
  |
  +-- var
       |
       +-- log
       |    |
       |    +-- nl_pipeline.log
       |    +-- nl_loader.log (*)
       |    +-- nl_parser.log (*)
       |
       +-- run
            |
            +-- nl_loader.pid
            +-- nl_loader.state (*)
            +-- nl_parser.pid
            +-- nl_parser.state (*)
            +-- nl_pipeline.pid

(*) Recommended, but not done by default: these must be
configured in nl_loader.conf/nl_parser.conf
```

5.4.1.1 Include syntax

The nl_loader and the nl_parser's output need to have some of the same configuration information: the nl_parser's output files are the nl_loader's input files; and it is convenient to store the current state and internal log files in the same place. For this reason, an "include" mechanism has been added that allows one configuration file to include another.

One possible way to use this mechanism is to place shared directory information in one file, e.g. `shared.conf`, and logging configuration in another, e.g. `logging.conf`. This layout is shown in the following figure.

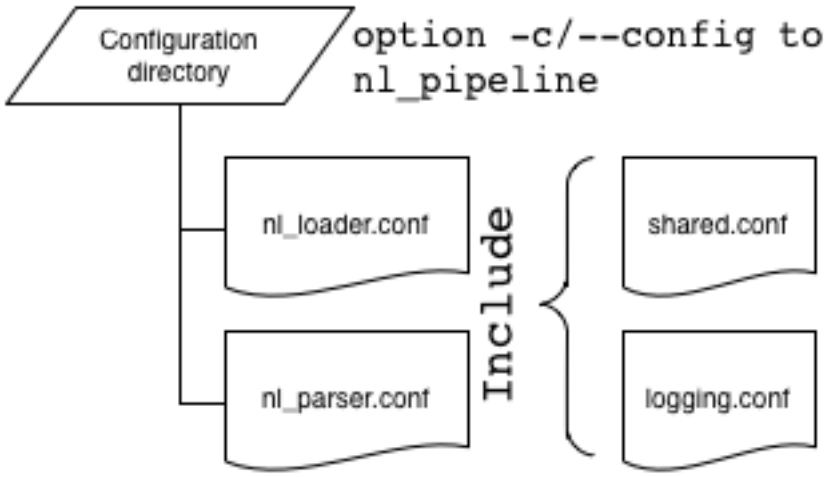

Figure 3: 3: nl_pipeline includes

To include, a file, simply put "@include" followed by a filename:

```
@include shared.conf
```

This literally reads the file in at that point. Multiple files can be included, but the included files cannot themselves have "@include" (i.e. no nested includes). Files which are included later can reference variables defined in files that were included earlier.

For example, in the layout above, let's say that shared.conf had the following:

Example 5.1 shared.conf

```
base_dir=/scratch/users/dang
```

Then the nl_loader.conf (and nl_parser.conf) file could put its "state" file in the recommended layout with the following:

Example 5.2 nl_loader.conf

```
# note: configuration file snippet, not complete
@include shared.conf
[global]
state_file = ${base_dir}/var/run
```

And, if the logging configuration was parameterized by using the variable `$prog` as the program name (i.e. "nl_loader" or "nl_parser"), then nl_loader.conf could define `$prog` appropriately and include the logging.conf as well.

Example 5.3 nl_loader.conf (2)

```
# note: configuration file snippet, not complete
@include shared.conf
[global]
state_file = ${base_dir}/var/run
prog = nl_loader
@include logging.conf
```

A more complete example is given at the end of this section.

5.4.1.2 Logging extensions

The internal logging of the nl_loader, nl_parser, and nl_pipeline is all done in the Best Practices (BP) format. However, configuration of the logging looks almost identical to the standard Python "logging" module's configuration file format. It is not quite the same, because it was considered useful to be able to include the logging configuration in the same file as the rest of the configuration parameters. This is the same model used by the TurboGears web framework.

In addition, the "loggers" instantiated at run-time by the programs are transparently wrapped to produce a BP-format log entry. By default, this is the entire output line; but the logging module allows user-defined formatters, and in this case the BP log ends up in the `%(message)s` format attribute.

By now, most non-Python programmers' heads are spinning. Before continuing, let's look at an example:

Example 5.4 Logging configuration

```
[logging]                                  ❶
[[loggers]]                                ❷
[[[netlogger]]]
level=DEBUG                                ❸
handlers=h1                                ❹
qualname=netlogger.nl_loader               ❺
propagate=0                                ❻
[[handlers]]
[[[h1]]]
class="FileHandler"                        ❼
args="('/var/log/nl_loader.log', 'a')"     ❽
```

Logging configuration annotations

❶ Top-level container for the logging configuration

❷ List of the logging.Logger instances to create

❸ Log all messages at severity DEBUG or higher.

❹ Handle messages with this (comma-separated list) of handlers

❺ All loggers whose name starts with "netlogger.nl_loader" should go here

❻ Do not log this message in less-specific loggers

❼ This handler logs to a file

❽ Append to file /var/log/nl_loader.log

For the most part, changing the level of logging for the whole program, or a specific part of the program, is a matter of simply adding a new sub-sub-section under `[[loggers]]` and setting the `qualname` and `level` variables to the desired part of the program and logging level. The following `qualname` values can be used:

netlogger All NetLogger program events

netlogger.nl_loader Startup/shutdown of the nl_loader

netlogger.nl_parser Startup/shutdown of the nl_parser netlogger.nl_pipeline All events in the nl_pipeline

netlogger.nlparser Internal events in the nl_parser

netlogger.nlloader Internal events in the nl_loader

5.4.1.3 Complete example

This example has three files: nl_parser.conf, nl_loader.conf, and logging.conf. It shows how the logging might be configured and re-used for a parser and loader.

Example 5.5 Parent file nl_parser.conf

```
# By defining the program name
# here and including the file logging.conf,
# it can be (re-)used for both the nl_parser
# and nl_loader.
prog=parser
@include logging.conf
```

Example 5.6 Parent file nl_loader.conf

```
prog=loader
@include logging.conf
```

Example 5.7 Child file logging.conf

```
\[logging]
[[loggers]]
# This logger is for events in the
# script nl_{whatever} itself, e.g.
# startup and shutdown events.
[[[program]]]
level=INFO
handlers=h1,h2
qualname=netlogger.nl_${prog}
propagate=0
# This logger is for the actual work
# of the program, e.g.: opening, parsing/loading,
# and closing files; saving state; etc.
[[[internals]]]
level=DEBUG
handlers=h1,h2
qualname=netlogger.nl${prog}
propagate=0
[[handlers]]
[[[h1]]]
class="FileHandler"
args="('$output_dir/${prog}.log', 'a')"
[[[h2]]]
# Only log errors
class="FileHandler"
level=ERROR
args="('$output_dir/${prog}-errors.log','a')"
```

5.4.2 Rotating files

If the NetLogger pipeline runs for a long time, the parsed files can grow large. In order to provide an orderly way of cleaning up old files, the nl_parser has a file rotation option. This can be controlled with a configuration file option, in the `[global]` section, called `rotate`. The value of the option is a time period given as the number of seconds, minutes, hours, or days; these can each be abbreviated with their first letter, but not combined.

Example 5.8 File rotation interval

```
# OK:
rotate = 1 # 1 second (default unit is seconds)
rotate = 1s # 1 second
rotate = 1 second # 1 second
rotate = 12 hours # 12 hours
rotate = 7 days # 1 week
# BAD:
rotate = 0.5 days # Only integers allowed
rotate = 1h 5m # Cannot combine time periods
rotate = 12 min # Use either 'm' or 'minutes'
```

When this parameter is specified to the nl_parser then its output file, let's say "ofile", has a numeric suffix ".<N>" appended to it, to make, for example, "ofile.1". Each time the file rotates, the nl_parser writes a special EOF event at the end of the current file, and opens a new file with the N+1 as its suffix, e.g., "ofile.2".

Running in a directory with existing output files
You don't need to worry about existing files: when it starts, the nl_parser automatically searches in the current directory for the highest numbered file that fits the pattern of "<ofile>.<N>" and starts at "<ofile>.<N+1>". It only does this once, so you shouldn't run two nl_parsers in the same directory; but this shouldn't be a problem in practice.

For its part, nl_loader is instructed to look for these numbered files by adding the boolean option `numbered_files` to its `[global]` section. For given value of <ofile> given to the `filename` option, it will automatically start reading the lowest-numbered file matching the pattern "<ofile>.<N>".

The nl_loader will continue to look for new data in this file until it reads the special EOF-event placed there by the nl_parser, and at that point it will do one of two things: rename the file or delete it (keeping the file name the same is not an option because on a restart of the nl_loader this would cause the same file to get loaded twice). The action is controlled by three options in the `[global]` section of the configuration file, as shown in the following snippet:

Example 5.9 nl_loader options for rotated (numbered) files

```
\[global]
numbered_files = yes
#=====================================
# Exactly one of the following options
# must be given
# =====================================
# To rename the output file with a new suffix
move_files_suffix = ".PARSED"
# To move the output file to another directory
move_files_dir = /scratch/parsed-files
# To delete the file entirely
delete_old_files = yes
```

The nl_loader will exit with an error if numbered_files is true and there is no option indicating how to deal with old files.

5.4.3 Sending messages to nl_loader and nl_parser

The nl_pipeline sends periodic messages to the nl_loader and nl_parser to tell them to save state (see sections on nl_parser state and nl_loader state), re-read configuration files, and also a message before it terminates so they can shut down cleanly. This is done with small UDP messages.

Explanation of the —secret option
So that not just anyone can send a UDP message to a running nl_parser, and nl_loader instance, there is a small file with a "secret", that is readable by the user who started the nl_pipeline process, that is used to encrypt the messages. The location and contents of this file is controlled by the nl_pipeline program, and then its location is passed to the nl_parser and nl_loader with the `--secret` command-line option.

The nl_parser is capable of re-reading its configuration while running. This is important because the set of logfiles matched against the wildcards for the `file` option in each parser section is determined at configure time. This means that if you have a section like the following, and you create a new file `/scratch/var/foobar.log`, the file will **not** be parsed until it is re-configured.

Example 5.10 Wildcard expression for nl_parser input files

```
\[global]
files_root = /scratch/var
[parser_section]
[[parser_module]]
files = *.log
```

Obviously, restarting the nl_parser will also reconfigure it, but a lighter weight alternative is to simply reconfigure it (this also avoids the problem of having the nl_pipeline wait, multiple times, for the nl_parser to enter a "good" state). To deal with this, the nl_pipeline `-i/--interval` option is used to determine the interval between sending messages to reconfigure. The time interval syntax is the same as for the file rotation interval.

Example 5.11 Send reconfigure message interval

```
nl_pipeline -i 5m -c /path/to/conf/etc ...
```

An authorized user can also trigger reconfiguration by hand by sending the nl_parser process a SIGHUP signal (e.g., `kill -SIGHUP <pid>`).

5.5 SQL and R Analysis

SQL queries can be used directly to perform some types of analyses. We have written a tool called nl_dbquery to package up a group of related SQL queries and allow some parameters to those queries to be passed as command-line options.

When more complicated numeric manipulations are required, we use the R language. See the section on installing R for details on how to get R, and the R libraries used below, installed on your system. For more details on the R language, consult the extensive built-in and online documentation.

5.5.1 Performing SQL queries from R

One of the nice features of R is that you can connect to the database directly with a pretty simple API. The data is returned in an R data structure called a "data frame", which is very much like a database table in that it consists of named and numbered columns, each with their own datatype.

- MySQL

```
# Load MySQL DB-API. You can also add this line
# to ~/.Rprofile so you don't need to type it # every time.
library(RMySQL)
# Connect to database using ~/.my.cnf
con <- dbConnect(MySQL())
# Choose database
dbGetQuery(con, paste("use", dbname))
# Run query, get result as a data frame
df <- dbGetQuery(con, "select count(*) from event")
```

- SQLite

```
# Load library
library(RSQLite)
# Connect to database file /path/to/db.sqlite
con <- dbConnect(SQLite(), dbname="/path/to/db.sqlite")
# Run query
df <- dbGetQuery(con, "select count(*) from event")
```

- PostgreSQL

```
# SORRY, still working on this one..
```

5.5.2 Creating plots with R

Not much in the way of pre-packaged general purpose software is available for this yet. Here is a quick example of how to draw a 2-D plot. Let's say you have about time and duration for some event, for example from the following query:

```
df <- dbGetQuery(con, "select event.time, attr.value from event join
attr on event.id = attr.e_id where event.name = 'pegasus.invocation'
and attr.name = 'duration' limit 100")
```

In this case the plotting function is *xyplot* from the *lattice* library.

```
# load plotting library
library(lattice)
# Convert duration to a numeric value (the datatype is VARCHAR in the DB)
df$value <- as.numeric(df$value)
# Create a new column, seconds since first timestamp
df$sec <- as.numeric(df$time) - min(as.numeric(df$time))
# Plot the data
xyplot(value ~ sec, data=df, xlab="Time (sec)", ylab="Duration (sec)", main="Sample plot")
```

The resulting plot would look something like this:

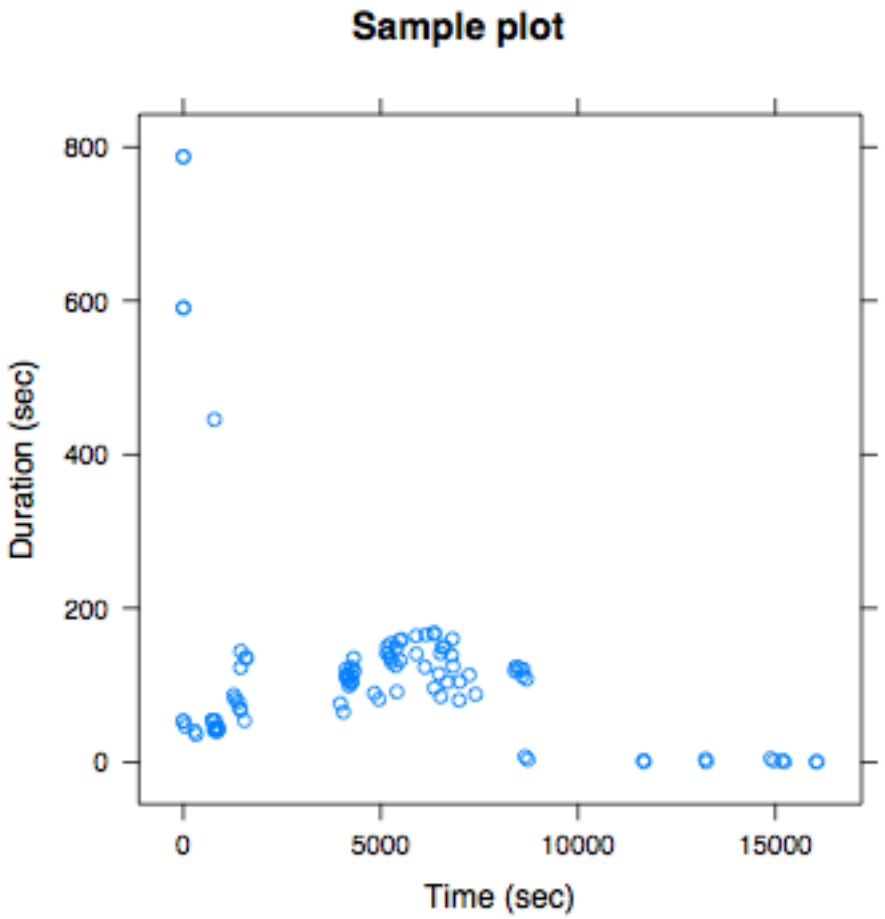

Figure 4: 4 Sample xyplot() output

6 Frequently Asked Questions

What is NetLogger? NetLogger is a methodology for troubleshooting and analyzing distributed application. The NetLogger Toolkit is a set of tools that help deploy this methodology. The methodology is described in more detail here .

Is the current version compatible with previous version(s)? In a word, no. NetLogger has been in existence, in one form or another, since 1994. Since that time it has been rewritten and renamed, so that the body of software now labeled NetLogger has little or no relation to the software distributed in the early years of research and development.

Why is it called NetLogger? NetLogger is short for "Networked Application Logger". NetLogger is NOT just about monitoring the Network.

Is NetLogger Open Source? Yes! It is under a BSD-style open source license.

What happened to the binary format, the activation service, and other features described in some of the NetLogger papers? They were not used by anyone, and so they were removed to make NetLogger smaller and easier to install.

Is NetLogger compatible with Java's logging package (aka log4j)? Yes.

What is the overhead of adding NetLogger? The overhead is very low. You can generate up to 5000 events/second using the C API, 500 events/second using the Java API, and 80 events/second using the python API which negligible impact on your application.

How do I analyze NetLogger log files? This is what the NetLogger Pipeline does. There is also a text-based viewer called "nl_view" that can make human browsing of the logs easier.

Questions we have not yet addressed? Please e-mail us at netlogger-dev@george.lbl.gov

A Tool manual pages

This section provides a version of the manpage documentation available, via the UNIX *man* command, for each of the tools in the NetLogger Python distribution.

A.1 netlogd(1)

A.1.1 NAME

netlogd - Receive logs over TCP or UDP and write them to a file.

A.1.2 SYNOPSIS

netlogd [options] [-h]

A.1.3 DESCRIPTION

The netlogd program combines one or more streams of newline-delimited log records into a single file. No checking is done as to the format of the records. Records are freely interleaved in a first-come, first-written manner. UDP and TCP mode cannot be used together.

A.1.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-b, --fork fork into the background after starting up

-f, --flush flush all outputs after each record

-k TIME, --kill=TIME Kill self after some time. Time can be given with units 's','m', or 'h' for seconds, minutes or hours. Default units are minutes ('m')

-o URL, --output=URL Output file(s), repeatable (default=stdout)

-p PORT, --port=PORT port number (default=14380)

-q, --quiet print nothing to stderr, overrides '-v'

-r SIZE, --rollover=SIZE roll over files at given file size (units allowed)

-U, --udp listen on a UDP instead of TCP socket

-v, --verbose verbose mode (report throughput)

A.1.5 EXAMPLES

To receive records on the default TCP port and write them to standard output:

```
$ netlogd
```

To receive records on UDP port 44351 and write them to file /tmp/combined.log:

```
$ netlogd -U -p 44351 -o /tmp/combined.log
```

A.1.6 EXIT STATUS

netlogd returns zero on success, non-zero on error

A.1.7 BUGS

None known.

A.1.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.2 nl_check(1)

A.2.1 NAME

nl_check - Check a log file for correctness.

A.2.2 SYNOPSIS

nl_check [options] [filename]

A.2.3 DESCRIPTION

Checks that a log file is formatted according to the CEDPS project "Best Practices" guide format (see RESOURCES).

Files are read from a list given on the command line or, if no files are listed, from standard input. Each line that does not conform is reported to standard output. Warnings and errors are printed to standard error, as well as the optional "progress" (useful for large files). In addition, the user may opt to make a copy of each input file with the offending lines removed (see -c option for details).

A.2.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-c, --clean write a copy of all 'clean' lines to stdout

-f, --fast Do a quick-and-dirty check

-p, --progress report progress to stderr

A.2.5 EXAMPLES

To print out errors in files a.log, b.log, and c.log to stdout:

```
nl_check a.log b.log c.log
```

To combine valid lines from files a.log, b.log, and c.log into cleaned.log, printing out errors to stderr:

```
nl_check -cx < a.log b.log c.log > cleaned.log
```

To check file big.log, copying valid lines to big.log.cleaned, showing progress (and validation errors) to stderr:

```
nl_check -p -c .cleaned big.log
```

A.2.6 EXIT STATUS

nl_check returns zero on success, non-zero on failure

A.2.7 BUGS

None known.

A.2.8 RESOURCES

BP Format - http://www.cedps.net/wiki/index.php/LoggingBestPractices

A.2.9 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.3 nl_check_pipeline(1)

A.3.1 NAME

nl_check_pipeline - Nagios-friendly script to check status of a NetLogger Pipeline

A.3.2 SYNOPSIS

nl_check_pipeline [options]

A.3.3 DESCRIPTION

nl_check_pipeline will check the status of your NetLogger pipeline by querying the process table via the *ps* command for the existance of processes named: *nl_parser*, *nl_pipeline* and *nl_loader*.

Additionally a check of the database will be done by connecting to it (see the -C option).

A.3.4 OPTIONS

-h, --help show this help message and exit

-v, --verbose verbosity, repeatable

-c FILE, --config=FILE Read configuration for this script from FILE. This will work with MySQL my.cnf files ([client] section). (default=~/.my.cnf)

A.3.5 EXAMPLES

Look for the running components: *nl_parser*, *nl_pipeline*, *nl_loader* and attemtp to connect the database as described in *~/.my.cnf*:

```
nl_check_pipeline
```

Same as above but display more information on missing components or problems connecting to the DB:

```
nl_check_pipeline -v
```

Here's a crontab entry using nl_check_pipeline in conjunction with nl_notify

```
# check nl_pipeline every 30 minutes
*/30 * * * * /opt/netlogger/bin/nl_notify -f dang@osp.nersc.gov -t dkgunter@lbl.gov -s  ↩
    smtp2.lbl.gov -g /opt/netlogger/bin/nl_check_pipeline
```

A.3.6 EXIT STATUS

0 Success

1 Duplicate components were found running.

2 Not all components were found running.

3 Other or unknown failure.

A.3.7 BUGS

Currently only supports MySQL database, needs to support PostgreSQL and SQLite.

Can be fooled by processes with identical names

A.3.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.4 nl_cpuprobe(1)

A.4.1 NAME

nl_cpuprobe - Measure CPU availability by active probing.

A.4.2 SYNOPSIS

nl_cpuprobe [options]

A.4.3 DESCRIPTION

Measure CPU availability by periodically spawning off a process that spins in a tight loop, and measuring the amount of the CPU we were able to get during that time. This should in theory be similar to the amount of resources a user application could claim.

For each probe, output is a line with a single floating-point number representing the estimated available CPU, in the range 0 to 1.

A.4.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-m MS, --millis=MS number of milliseconds out of every second to run the probe (default=100)

-n NICE, --nice=NICE nice value to give to the process while probing (default=0)

A.4.5 EXAMPLES

To run with spin-interval 50ms and nice value of 0:

```
$ nl_cpuprobe -m 50
```

To run, as root, with spin-interval 100ms and nice value of -5:

```
$ sudo nl_cpuprobe -m 100 -n -5
```

A.4.6 EXIT STATUS

nl_cpuprobe returns zero on success, non-zero on error

A.4.7 BUGS

None known.

A.4.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.5 nl_date(1)

A.5.1 NAME

nl_date - Convert floating-point dates to NetLogger string dates, and vice-versa

A.5.2 SYNOPSIS

nl_date [options]

A.5.3 DESCRIPTION

This utility just converts one or more dates from the number of seconds since the Epoch (1/1/1970 00:00:00) to the ISO8601 string representation YYYY-MM-DDThh:mm:ss.ffffffZ, or vice-versa. The type of a given input is auto-detected. NetLogger's own parsing and formatting routines are used, so this utility doubles as a sanity-check of those functions.

The date to convert is read from the command line, and output is printed to standard output in the form: "`input => output`". If no date is provided, then the output shows the current date in both formats, with the prefix "`now => `".

A.5.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-d DATE, --date=DATE convert the given date, which may be a fuzzy date like 'now', 'yesterday', or '1 week ago', as well as YYYY- MM-DD notation

-s SEC, --seconds=SEC convert from this number of seconds since the UNIX epoch (1970-01-01T00:00:00Z)

-z, --utc interpret given date or default 'now' as being in UTC instead of the local timezone

A.5.5 EXAMPLES

To print out the current date in both formats:

```
$ nl_date
now => 2008-09-24T20:17:40.594915-08:00 => 1222316260.594915
```

To convert a floating-point date to a string:

```
$ nl_date -s 1185733072.567627
1185733072.567627 => 2007-07-29T18:17:52.567627Z
```

To convert a string date to a floating-point date:

```
$ nl_date -d 2007-07-29T18:17:52.567627Z
2007-07-29T18:17:52.567627Z => 1185733072.567627
```

A.5.6 EXIT STATUS

nl_date always returns zero (success). If the arguments are not understood it just prints the current date.

A.5.7 BUGS

None known.

A.5.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
.........................................................
```

A.6 nl_dbquery(1)

A.6.1 NAME

nl_dbquery - A script for running user defined queries against an SQL database (generated by netlogger or otherwise).

A.6.2 SYNOPSIS

nl_dbquery [options]

A.6.3 DESCRIPTION

The query file takes various command line parameters as user input and queries the database according to the user defined queries and input parameters. The SQL queries to be executed are read from a config file. The SQL in the config file can contain parameters, which are filled in by the user when the tool is invoked. Special support is provided for the time range parameter.

A.6.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-c FILE, --config=FILE Read configuration from FILE. Default=./nl_dbquery.conf.The configuration file should follow the syntax in this file for it to be successfully parsed by the script. The configuration file is forreading the query information and other parameters from a database.Its also used for automatically generating a part of the help message itself.

-d DB_NAME, --db=DB_NAME Database to connect to. Default=pegasus

-l, --list This generates a numbered list of the availablequeries reading the query information from the configfile

-n, --dry-run Display but don't run the query

-p DB_PARAM, --param=DB_PARAM Database connection parameters (full path to the filename in case of a sqlite database or a host name in case of MySql). The host name for MySQL should be of the form mysql://hostname while for SQLite it should be of the form sqlite:///path/to/filename. This parameter is a required parameterexcept in case when the script is executed just with the -l/--list option to list the available queries.

-P QUERY_PARAM, --query-param=QUERY_PARAM Parameter for the given query, in the form 'name=value'. The 'value' is substituted for occurrences of '<name>' in the query string. May be repeated.

-q QUERY, --query=QUERY Run QUERY, which can be a number or name. Use -l/--list to list available queries

-u URI, --uri=URI Database connection URI, where the database module name is used as the URI scheme. MySQL requires a host and sqlite requires a filename.

-v, --verbose Repeat up to 3 times for more verbose logging. The default level is ERROR

A.6.4.1 Time Range:

The following start/end times for the query accept many date expressions like 'yesterday', '2 weeks 1 day ago', 'last wed', 'Jan 4', etc. A more complete list is: <N> weeks|days|hours|minutes|seconds ago, today, now, tomorrow, yesterday, 4th [[Jan] 2003], Jan 4th 2003, mm/dd/yyyy (preferred), dd/mm/yyyy, yyyy-mm-dd, yyyymmdd, next Tuesday, last Tuesday $$$$

-s START, --start=START Start date for the query ['1 week ago'].

-e END, --end=END End date for the query ['today'].

A.6.5 USAGE

To run, provide the configuration file location, the name or number of the query, the parameters for the query, and the connection information for the database. The results will be printed to standard output. You can also list the available queries, or do a dry run that just shows the actual SQL that would have been sent to the database.

The configuration file uses the INI format, with one square-bracketed [[section]] for each query. The name of the section is one way that the user can select the query, so make it short and informative. Inside each section two values are defined, *desc* for a description of the purpose and result of the query and *query* for the SQL of the query. Parameters in the configuration file are enclosed in angle brackets "<like_this>". Parameter names should not contain whitespace. The parameter "<timerange>", for the time range of the query, is reserved.

Parameters on the command-line are given with the **-P/—query-param** option, except in the case of the timerange which uses the **-s/—start** and **-e/—end** options.

A.6.6 EXAMPLES

Here is an example configuration file, with 2 queries in it. Note that use of triple-quotes allows prettier formatting of the query.

```
[how_many_jobs]
desc = "How many jobs ran on a given day"
query = "select count(id) from event where <timerange> and name = 'pegasus.invocation';"
[job_status]
desc = "How many jobs had a given status"
query = """
   select count(id)
   from
       attr join event on e_id = id
    where
       <timerange> and event.name = 'pegasus.invocation'
       and attr.name = 'status' and attr.value = '<status>' """
```

To list available queries:

```
$ nl_dbquery -c my.conf -l
```

To run a query with a timerange:

```
$ nl_dbquery -c pegasus.conf -q how_many_jobs -u mysql://localhost \
  -s 2008-01-01 -e 2009-01-01 -d usc1
# OUTPUT:
Running query: select count(id) from event where time >=
unix_timestamp('2008-01-01') and time <= unix_timestamp('2009-01-01')
and name = 'pegasus.invocation';

001: 41616

Query execution time: 0.082433 seconds
```

To run a query with a timerange and an additional parameter:

```
$ nl_dbquery -c pegasus.conf -q job_status -u mysql://localhost \
  -s 2008-01-01 -e 2009-01-01 -d usc1 -P status=0
```

A.6.7 EXIT STATUS

nl_dbquery returns 0 on success and a non-zero value on failure.

A.6.8 BUGS

None known.

A.6.9 AUTHOR

Binit Bhatia <bsbhatia@lbl.gov>

Dan Gunter <dkgunter@lbl.gov>

Keith Beattie <ksbeattie@lbl.gov>

```
.........................................................
```

A.7 nl_findbottleneck(1)

A.7.1 NAME

nl_findbottleneck - Find bottleneck from NetLogger transfer summary logs.

A.7.2 Synopsis

nl_findbottleneck [options] [log-file]

A.7.3 DESCRIPTION

Determine the bottleneck from NetLogger logs that show the disk and network read and write bandwidths. The input is a NetLogger log, specifically the one produced by NetLogger's "transfer" API, although in reality the only fields that need to be present are the correct event name (see below) and:

r.s: sum of bytes/sec ratio

nv: number of items in the sum for **r.s**

The event name is expected to contain one of four values indicating the component being measured; "disk.read", "disk.write", "net.read", and "net.write". As long as this string appears somewhere in the event name, it will be recognized.

The output is the bottleneck, or "unknown". Optionally (with **-v**), the sorted list of bandwidths is written as well.

Although the options provide for multiple bottleneck algorithms, at present only one is implemented — the "simple" algorithm that basically looks for the smallest number and labels that the bottleneck if it is more than 15% smaller than the next smallest. For details see the netlogger.analysis.bottleneck module.

Note that parse errors in the input files will be silently ignored. If the **-d** flag is given, then parse errors will show up as debug messages in the log, but they still will not stop the program.

A.7.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-d, --debug log debugging information, including parsing errors

-a ALG, --algorithm=ALG choose bottleneck algorithm by name (default=simple)

-v, --verbose verbose output mode (report all values)

A.7.5 EXAMPLES

To determine the bottleneck from my_transfer.log:

```
nl_findbottleneck my_transfer.log
```

A.7.6 EXIT STATUS

nl_findbottleneck returns zero on success, and non-zero on error

A.7.7 BUGS

None known.

A.7.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.8 nl_findmissing(1)

A.8.1 NAME

nl_findmissing - Find and display "missing" events in NetLogger (CEDPS Best-Practices format) logs.

A.8.2 SYNOPSIS

nl_findmissing [options] [files..]

A.8.3 DESCRIPTION

Read NetLogger logs as input and produce as output any .start/.end events that are missing their matching event. The user specifies what fields of a logged event are used for comparison, and this is even flexible enough to even allow different event names to be matched to each other.

Logs are read from standard input or a file, and output is written to standard output. Input lines in the logfile that are not understood, are silently ignored.

The **-i/—ids** option can be used to specify which fields should be used to match a starting event with its ending event. Optionally, a pattern can be placed before a ":" to filter the events that are being considered at all. If this option is not provided, then all events are considered and the fields *event* and *guid* (i.e. as if the user specified "-i event,guid") are used to match starting and ending events. This option may be repeated, so that different sets of events can use different sets of identifiers.

There are three output formats (see EXAMPLES):

- Human-readable
- Comma-separated values (CSV)
- Best Practices logging format (BP)

A.8.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-i IDS, --ids=IDS Set of identifying fields for a given event pattern, using the syntax: [EVENT_REGEX:]FIELD1,..,FIELDN (default='guid')

-t FMT, --type=FMT Output type (default=human)

-p, --progress report progress to stderr

A.8.5 EXAMPLES

To process the logs and produce human-readable output:

```
$ nl_findmissing  -t human log2
log2: lala.13 missing end
log2: po.34 missing end
```

To process the logs and produce CSV output:

```
$ nl_findmissing  -t csv log2
file,event,missing,key
log2,lala.13,end,lala.13/A2C4144D-7684-FA3E-8F5B-F0E34D8BC18E
log2,po.34,end,po.34/6275D71E-D023-A9F6-742E-6512DD90A1F1
```

To process the logs and produce BPoutput:

```
$ nl_findmissing  -t log log2
ts=2008-09-25T18:42:13.635438Z event=lala.13.start level=Info
guid=A2C4144D-7684-FA3E-8F5B-F0E34D8BC18E nl.missing=end mode=random
file=log2 guid=b09f6896-8b41-11dd-964e-001b63926e0d
ts=2008-09-25T18:42:13.635929Z event=po.34.start level=Info
guid=6275D71E-D023-A9F6-742E-6512DD90A1F1 nl.missing=end mode=random
file=log2 p.guid=A2C4144D-7684-FA3E-8F5B-F0E34D8BC18E
guid=b09f6896-8b41-11dd-964e-001b63926e0d
```

To match events starting with *airplane* on attributes *flightno* and *airline*, and all other events on a combination of *country* and *city*:

```
nl_findmissing -t log -i airplane:flightno,airplane -i country,city in.log > out.log
```

A.8.6 EXIT STATUS

nl_findmissing returns zero on success, non-zero on failure

A.8.7 BUGS

None known.

A.8.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
.....................................................
```

A.9 nl_ganglia(1)

A.9.1 NAME

nl_ganglia - Read Ganglia in, write NetLogger out

A.9.2 SYNOPSIS

nl_ganglia [options]

A.9.3 DESCRIPTION

Contact a Ganglia gmetad, parse the returned XML document, and convert the information into NetLogger-formatted output, with one log entry per metric.

A.9.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-e REGEX, --filter=REGEX regular expression to use as a filter. This expression operates on the formatted output, i.e. name=value pairs

-i SEC, --interval=SEC poll interval in seconds (default=run once)

-m METRICS, --metrics=METRICS set of metrics to display (default=base)

-o FILE, --output=FILE output file (default=stdout)

-s SERVER, --server=SERVER gmetad server host (default=localhost)

-p PORT, --port=PORT gmetad server port (default=8651)

-v, --verbose verbose output

A.9.5 EXAMPLES

To contact ganglia on default port and dump one set of default metrics to the console:

```
nl_ganglia
```

To contact ganglia on server *foobar.org* once every 15 seconds, and write the subset of returned metrics that contains *cpu* in the event name to the file */tmp/ganglia.out*:

```
nl_ganglia -e event='.*cpu' -s foobar.org -o /tmp/ganglia.out -i 15
```

A.9.6 EXIT STATUS

nl_ganglia returns zero on success, non-zero on failure

A.9.7 BUGS

None known.

A.9.8 RESOURCES

Ganglia Monitoring System - http://ganglia.info

A.9.9 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.10 nl_interval(1)

A.10.1 NAME

nl_interval - Read NetLogger logs as input and output the interval between the .start and .end events.

A.10.2 SYNOPSIS

nl_interval [options] [files..]

A.10.3 DESCRIPTION

Read NetLogger logs as input and produce as output intervals between .start/.end events. The user specifies what fields of a logged event are used for comparison, and this is even flexible enough to even allow different event names to be matched to each other.

Logs are read from standard input or a file, and output is written to standard output. Input lines in the logfile that are not understood, are silently ignored.

The **-i/—ids** option can be used to specify which fields should be used to match a starting event with its ending event. Optionally, a pattern can be placed before a ":" to filter the events that are being considered at all. If this option is not provided, then all events are considered and the fields *event* and *guid* (i.e. as if the user specified "-i event,guid") are used to match starting and ending events. This option may be repeated, so that different sets of events can use different sets of identifiers.

There are three output formats (see EXAMPLES):

- Human-readable
- Comma-separated values (CSV)
- Best Practices logging format (BP)

A.10.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-i IDS, --ids=IDS Set of identifying fields for a given event pattern, using the syntax: [EVENT_REGEX:]FIELD1,..,FIELDN (default='guid')

-p, --progress report progress to stderr

-t FMT, --type=FMT Output type (default=human)

A.10.5 EXAMPLES

Process *in.log* and produce human-readable output:

```
$ nl_interval < in.log
lala.24 0.000059
po.13 0.000041
tinkywinky.81 0.000039
tinkywinky.55 0.000042
```

Process *in.log* and produce CSV output:

```
$ nl_interval -t csv < in.log
event,key,interval_sec
lala.24,lala.24/C24391AA-4D28-78B1-D59C-9C96627F256F,0.000059
po.13,po.13/1C746366-6C8A-3238-7CF2-313C417ECF96,0.000041
tinkywinky.81,tinkywinky.81/31A15BAD-4AEE-1E63-7ACD-C6EB8CF8547B,0.000039
tinkywinky.55,tinkywinky.55/9A16401D-5643-69BF-DFE9-A95692A349A4,0.000042
```

Process *in.log* and produce log output:

```
$ nl_interval -t log < in.log
ts=2008-09-25T18:42:13.636326Z event=lala.24.intvl level=Info status=0
guid=C24391AA-4D28-78B1-D59C-9C96627F256F nl.intvl=0.000059
mode=random p.guid=6275D71E-D023-A9F6-742E-6512DD90A1F1
ts=2008-09-25T18:42:13.636653Z event=po.13.intvl level=Info status=0
guid=1C746366-6C8A-3238-7CF2-313C417ECF96 nl.intvl=0.000041
mode=random p.guid=6275D71E-D023-A9F6-742E-6512DD90A1F1
ts=2008-09-25T18:42:13.636927Z event=tinkywinky.81.intvl level=Info
status=-1 guid=31A15BAD-4AEE-1E63-7ACD-C6EB8CF8547B nl.intvl=0.000039
mode=random p.guid=6275D71E-D023-A9F6-742E-6512DD90A1F1
ts=2008-09-25T18:42:13.637220Z event=tinkywinky.55.intvl level=Info
status=0 guid=9A16401D-5643-69BF-DFE9-A95692A349A4 nl.intvl=0.000042
mode=random p.guid=6275D71E-D023-A9F6-742E-6512DD90A1F1
```

Match events starting with *airplane* on *flightno* and all other events on a combination of *country* and *city*.

```
nl_interval -i airplane:flightno -i country,city < in.log > out.log
```

A.10.6 EXIT STATUS

nl_interval returns zero on success and non-zero on failure

A.10.7 BUGS

None known.

A.10.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
.....................................................
```

A.11 nl_loader(1)

A.11.1 NAME

nl_loader - Load NetLogger log files into a relational database

A.11.2 SYNOPSIS

nl_loader [options]

A.11.3 DESCRIPTION

Load NetLogger log files into a relational database, using a fixed general-purpose schema. The supported databases are SQLite, PostgreSQL, MySQL, and a "test" database that simply prints the SQL statements to the console.

A.11.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

A.11.4.1 Pipeline-mode options:

-c FILE, --config=FILE use configuration in FILE

-d PIDFILE, --daemon=PIDFILE run in daemon mode, writing PID to PIDFILE. Requires -c, --config option.

-n, --no-action do not do anything, just show what would be done (default=False)

A.11.4.2 Standalone-mode options:

-b IBATCH, --insert-batch=IBATCH number of INSERT statements per transaction (default=100)

-C, --create create new database before using it

-D, --drop drop existing database first; implies -C/--create

-i FILE, --input=FILE read input from FILE (default=stdin)

-p DB_PARAM, --param=DB_PARAM parameters for database connect() function, formatted as 'name=value'. repeatable. For MySQL, use read_default_file=~/.my.cnf to avoid putting the password on the command-line.

-r FILE, --restore=FILE restore log file name and offset from FILE

-s FILE, --schema-file=FILE Read schema configuration from FILE

--secret=FILE For pipeline mode, FILE has authentication secret to use with commands with nl_pipeline. Users should not need to worry about this option.

--schema-init=KEY 1,KEY2, .. Comma-separated keys for type of initialization schema to use (default=first in file)

--schema-finalize=KEY 1,KEY2, .. Comma-separated keys for type of finalization schema to use (default=first in file)

-u URI, --uri=URI database connection URI, where the database module name is used as the URI scheme. MySQL and PostgreSQL modules require a server host; the sqlite and test modules require a filename. (required)

-U, --no-unique do not enforce unique events. With -C or -D, this removes the UNIQUE constraint on the table. With existing tables, new keys are guaranteed unique regardless of the event.

-v, --verbose Verbose logging

A.11.5 USAGE

The program runs in two modes: *standalone* and *pipeline*. In standalone mode, you provide a list of files to load, and provide the connection URI, database parameters, etc. via command-line options. In *pipeline* mode, you are assumed to be running the **nl_loader** program to create a series of output files that the **nl_loader** is then loading into the database.

Pipeline mode uses a configuration file, specified with the **-c/—config** option. With this option, only the **-d/—daemon** option indicating whether to run in the background, and the **-n/—no-action** option saying whether to do a dry run, are honored. The rest of the command-line options have an equivalent in the configuration file. For details of the directives in this configuration file see the NetLogger manual, which can be found at NetLogger home page - http://acs.lbl.gov/NetLoggerWiki.

In standalone mode, the **-u/—uri** option is required. The input file can either be read from standard input, given explicitly with the **-i/—input** option, or inferred from the restore-state file given with the **-r/—restore** option.

For the **-p/—param** option, please note that the name=value pairs used as parameters to the connection are not standard across database modules. You will need to consult the documentation for the appropriate database and/or its python module. The modules used are as follows. sqlite: **sqlite3** for Python2.5+, **pysqlite2.dbapi2** for Python2.4 or lower; PostgreSQL: **psycopg2**, or **pgdb** if that's not found; MySQL: **MySQLdb**

For the **-r/—restore** option, note that the file need not exist, in which case it will be created. Subsequent invocations with the same file name will simply start where the last one left off. This removes the fear from loading very large files into the database, since you can interrupt with control-C and resume (with the same command line) multiple times.

A.11.6 EXAMPLES

To connect, in standalone mode, to MySQL on localhost, using database *nltest* as user *joe* with password *foobar*, and load in the data in *file.log*:

```
$ nl_loader -u mysql://localhost -p user=joe -p passwd=foobar \
            -p db=nltest -i file.log
```

Same as above, but use the MySQL configuration in *~/.my.cnf*, which contains the user, password, and database:

```
nl_loader -u mysql://localhost -i file.log
```

To parse the configuration file and report errors:

```
nl_loader -c my.conf -n
```

A.11.7 EXIT STATUS

nl_loader returns zero on success, non-zero on failure.

A.11.8 BUGS

None known.

A.11.9 RESOURCES

ConfigObj home page - http://www.voidspace.org.uk/python/configobj.html

A.11.10 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
. . . . . .
```

A.12 nl_notify(1)

A.12.1 NAME

nl_notify - Run a command and notify by email if it fails.

A.12.2 SYNOPSIS

nl_notify [options] command args..

A.12.3 DESCRIPTION

Runs a given command with its arguments. If return status from the command is non-zero, send the standard output and standard error, with an appropriate subject line, to the provided email address. If the return status from the command is zero, do nothing.

Email is sent by default to localhost, port 25. Values for the "From:" and "To:" fields must be provided by the user.

Note: If the command's arguments include a dash then they need to be quoted (you can quote the whole command if you want).

A.12.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-b SUBJECT, --subject=SUBJECT Email subject (default=Error on %host from '%prog')

-f user@host, --from=user@host Set 'From:' to user@host (required)

-g, --nagios Nagios mode. Put first line of standard output in '%status'. Add this to default subject line (default=No)

-n, --test Print to stdout instead of sending email

-p SERVER_PORT, --port=SERVER_PORT SMTP server port (default=25)

-s HOST, --server=HOST SMTP server host (default=localhost)

-t user@host, --to=user@host Set 'To:' to user@host (required)

A.12.5 EXAMPLES

To write what *would* have happened to standard output:

```
$ nl_notify --from user@somehost.com --to user@otherhost.org  --test /usr/bin/false
Connect to localhost:25
To: user@otherhost.org
From: user@somehost.com
Subject: Error on 192.168.1.101 (Macintosh-8.local) from '/usr/bin/false'
Output from '/usr/bin/false':
-- stdout --

-- stderr --
```

To run nl_check_pipeline in "nagios mode", which allows you to include the status in the subject line:

```
$ nl_notify -b "Hey: %host says \'%status\'" \
  -f user@somehost.org -t user@otherhost.com \
  -g -p 9999 nl_check_pipeline
Subject: Hey: 192.168.1.101 (Macintosh-8.local) says 'CRITICAL: 3 components not running'
Output from '../../scripts/nagios/nl_check_pipeline':
-- stdout --
CRITICAL: 3 components not running

-- stderr --
```

A.12.6 EXIT STATUS

nl_notify returns zero on success, nonzero on an error.

A.12.7 BUGS

None known.

A.12.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.13 nl_parser(1)

A.13.1 NAME

nl_parser - Parse various log formats and output NetLogger format

A.13.2 SYNOPSIS

nl_parser [options] files. . .

A.13.3 DESCRIPTION

This program converts from known log formats to NetLogger (a.k.a. CEDPS Best-Practices) format, which can then be used by the rest of the NetLogger tools. There are a number of built-in parsers. Any Python module implementing the API documented below can also be used as a parser. nl_parser can operate on many different files at once, using pattern rules to match parsers to files. It can also handle combined logs from different applications — e.g., as from syslog — as long as there is a header that can be used to distinguish them. The output is always a single file, standard output by default; see the Output section below for details.

This program can either run with command-line options or using a configuration file. It can also run as a daemon, which requires the use of a configuration file. Its functionality is restricted from the command-line to processing with a single parser module and writing to standard output.

A.13.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

A.13.4.1 General options:

-c FILE, --config=FILE use configuration in FILE

-d PIDFILE, --daemon=PIDFILE run in daemon mode, writing PID to PIDFILE.

-D PARSER_MODULE, --desc=PARSER_MODULE describe parser PARSER_MODULE

-g NUM, --progress=NUM report progress to standard error in increments of NUM lines

--secret=FILE For pipeline mode, FILE has authentication secret to use with commands with nl_pipeline. Users should not need to worry about this option.

A.13.4.2 Command-line parser configuration:

-e EXPR, --header=EXPR extract header from each line using EXPR, a regular expression

-m PARSER_MODULE, --module=PARSER_MODULE parser module to use if not using -c

-n, --no-action do not do anything, just show what would be done (default=False)

-p MOD_PARAM, --param=MOD_PARAM parameter for module if not using -c, in the format name=value. Repeatable. Any parameter not used by the parsing module will end up as a name=value appended to each line of output.

-t THROTTLE, --throttle=THROTTLE maximum fraction of CPU to use (default=1.0)

-u FILE, --unparsed-file=FILE file to store unparseable events in (default=none)

-v, --verbose Verbose logging

-x, --external Module is external, not in netlogger.parsers.modules. Look first in '.'

A.13.5 USAGE

The program runs in two modes: *standalone* and *pipeline*. In standalone mode, a parser is applied to a list of files. In *pipeline* mode, a configuration file specifies the mapping of a number of parsers to a number of files, as well as some additional options to tail input and rollover output.

In addition, the **-D/—desc** option can be used to describe the parameters taken by a given parser. The **-n/—no-action** option can be used to parse (and thus check) a configuration file without actually running. All the command-line options in the section titled "Command-line parser configuration" have an equivalent in the configuration file.

For details on the directives in this configuration file see the NetLogger manual, which can be found at NetLogger home page - http://acs.lbl.gov/NetLoggerWiki.

To see how to write your own parser modules, see the NetLogger cookbook, which can also be found in the documentation area under NetLogger home page - http://acs.lbl.gov/NetLoggerWiki.

A.13.6 SIGNALS

Some signals cause nl_parser to perform special actions:

- SIGTERM, SIGINT, SIGUSR2: Terminate gracefully
- SIGUSR1: Rotate the output file. The current file's contents will be moved to a new name with a unique prefix in the same directory, and the file will start over at zero length.
- SIGHUP: Re-read the configuration file. This results in all the input files being closed and re-opened, although assuming persistence is turned on this should not cause any anomalies in the processing of files that are present in both configurations.

A.13.7 EXAMPLES

To parse a single file:

```
$ nl_parser --module=bp sample.log > sample-parsed.log
```

To parse multiple files:

```
$ nl_parser --module=bp sample1.log sample2.log > sample-parsed.log
```

To strip a header from a file:

```
$ nl_parser --module=bp --expr="\S+: " sample.log > sample-parsed.log
```

A.13.8 EXIT STATUS

Returns zero on success, non-zero when it encounters a misconfiguration, missing file, or fatal parsing error.

A.13.9 BUGS

None known.

A.13.10 RESOURCES

ConfigObj home page - http://www.voidspace.org.uk/python/configobj.html

A.13.11 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.14 nl_pipeline(1)

A.14.1 NAME

nl_pipeline - Program to manage the nl_parser and nl_loader

A.14.2 SYNOPSIS

nl_pipeline [options]

A.14.3 DESCRIPTION

Program to fork and manage the nl_parser and nl_loader.

The **nl_pipeline** starts both programs as daemons. Then periodically sends the **nl_parser** a message so it re-reads its configuration file and, more importantly, picks up on input files that have come or gone since the last configuration.

For testing and debugging, you can use the **-n/—no-action** option, which tells the program to just check that it could have started both the nl_parser and the nl_loader, and report how. For debugging while running, you can also set the logging verbosity with **-v/—verbose** and run in the foreground with **-D/—no-daemonize**

By default, the **nl_parser** and **nl_loader** programs are looked for in the same directory as the **nl_pipeline**; this is the normal situation if you installed them together on your system. To look in your PATH for the appropriate parser and loader scripts, use the **-s/—sys-path** option.

A.14.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-c DIR, --config-dir=DIR read nl_loader.conf and nl_parser.conf from DIR (required)

-D, --no-daemonize don't daemonize self, for debugging (default=False)

-i TIME, --interval=TIME interval for sending a signal to the parser to force it to re-read its configuration, and thus to notice new or deleted log files (30 seconds)

-l DIR, --log-dir=DIR write logs to DIR/nl_pipeline.log (default=CONF_DIR/../var/log)

-n, --no-action do not do anything, just show what would be done (default=False)

-p DIR, --pid-dir=DIR write nl_loader.pid and nl_parser.pid in DIR (default=CONF_DIR/../var/run)

-s, --sys-path search system path for nl_loader and nl_parser executables (default=False; look only in full path to nl_pipeline)

-w SEC, --wait=SEC wait up to SEC seconds for parser and loader to start (default=5)

-v, --verbose Verbose logging

A.14.5 EXAMPLES

To run with configuration files *~/local/netlogger/etc/nl_loader.conf* and *~/local/netlogger/etc/nl_parser.conf*, and place PIDs in *~/local/netlogger/var/run/nl_loader.pid* and *~/local/netlogger/var/run/nl_parser.pid*, and log internal status to *~/local/netlogger/-var/log/nl_pipeline.log*, do the following:

```
nl_pipeline -c ~/local/netlogger/etc
```

To use the same configuration as the above without really running (to test the configuration files):

```
nl_pipeline -c ~/local/netlogger/etc -n
```

To run in the foreground with logging turned up, with all files in /tmp/xyz:

```
nl_pipeline -c /tmp/xyz  -p /tmp/xyz -l /tmp/xyz -v -v -D
```

A.14.6 EXIT STATUS

Returns zero on success, 255 on failure

A.14.7 BUGS

None known.

A.14.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
. . . . . .
```

A.15 nl_view(1)

A.15.1 NAME

nl_view - Re-format NetLogger logs.

A.15.2 SYNOPSIS

nl_view [options] [files..]

A.15.3 DESCRIPTION

Reformats the semi-structured keyword and value pairs of the NetLogger format for readability or importing into Excel, R, or other programs that require tabular data.

The time and event is always shown, although the time can be formatted either as an absolute ISO timestamp (the default), or as a number of seconds since the first or previous event. An arbitrary prefix can be stripped from event names (names without that prefix are of course left alone).

The default delimiter between columns is a space, but this can be changed to make, e.g., comma-separated values. Currently no quoting is done.

Special support for identifiers is provided with the -t/—tiny-id option, which replaces the value of the identifier with a short (4-character) locally unique value. This value is random, but the seed is always the same and the algorithm is deterministic, so the chosen value will be the same for successive invocations.

A.15.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-a ATTR, --attr=ATTR add attribute ATTR to output line, repeatable

-A, --all add all attributes to output line

-c, --cum-delta show times as deltas since first (defalt=False)

-d, --delta show times as deltas from previous (defalt=False)

-D DELIM, --delimiter=DELIM column delimiter (default=' ')

-g, --guid add 'guid' attribute

-H, --header add header row (default=False)

-i, --host add 'host' attribute

-I, --identifiers add any attribute ending in '.id'

-l, --level add 'level' attribute'

-m add 'msg' attribute

-n PREFIX, --namespace=PREFIX strip namespace PREFIX if found

-N, --no-names Do not show attribute names

-s, --status add 'status' attribute

-t, --tiny-id replace *.id and guid values with shorter id's, like tinyurl

-w NUM, --width=NUM set event column width to NUM (default=40)

-x ignore non-NetLogger lines

A.15.5 EXAMPLES

To put the viewer in a pipeline between the application and a pager:

```
my-application | nl_view -gi | less
```

To run the viewer on a bunch of files, showing some user-defined attributes:

```
nl_view -a foo -a bar *.log > combined.log
```

To run the viewer so that it displays time-deltas, guid, event name with a prefix stripped, and any "identifier" attributes (this particular set of values is useful for the Globus 4.2 containerLog):

```
nl_view -diIgmt --namespace=org.globus.  containerLog
```

A.15.6 EXIT STATUS

Always succeeds, returning 0.

A.15.7 BUGS

None known.

A.15.8 RESOURCES

Apache Common Log Format - http://httpd.apache.org/docs/2.2/logs.html

A.15.9 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

```
. . . . . .
```

A.16 nl_wflowgen(1)

A.16.1 NAME

nl_wflowgen - Generate simulated workflow logs.

A.16.2 SYNOPSIS

nl_wflowgen [options] [-h]

A.16.3 DESCRIPTION

Generate random workflow logs in BP (NetLogger) format.

Two distinct types of simulated workflows can be generated. The *random* workflow is simply a random tree of events, linked together with GUIDs. The *globus* type workflow is not entirely like the logs from a Globus (GT4.2+) job submission.

How deeply workflows are nested is determined by the **—mindepth** and **—maxdepth** options, whereas the probability that the next event in any given workflow will be nested (if allowed by the min/max depth) is controlled by the **—nest** option.

Each ending event for a workflow has an associated *status* attribute. The probability of that being non-zero, i.e. indicating failure, is controlled with the **—fail** option.

A.16.4 OPTIONS

--version show program's version number and exit

-h, --help show this help message and exit

-m MODE, --mode=MODE Run mode (default=random). Modes: 'random' = a random workflow 'tree'; 'globus' = Globus job submit

-o OFILE, --output=OFILE output filename. use stdout if not given

--num=NUM [random, globus] number of events, total (default=100)

--mindepth=MIN_DEPTH [random] minimum number of nested events in a workflow (default=1)

--maxdepth=MAX_DEPTH [random] maximum number of nested events in a workflow (default=5)

--fail=FAIL [random] probability of failure for a .end event (default=0.1)

--nest=NEST [random] probability of nesting events, at any point (default=0.5)

A.16.5 EXAMPLES

To produce a default *random* workflow to standard output:

```
nl_wflowgen
```

To produce a default *globus* workflow to standard output:

```
nl_wflowgen -m globus
```

A.16.6 EXIT STATUS

Returns zero on success, non-zero on error

A.16.7 BUGS

None known.

A.16.8 RESOURCES

The Globus Alliance - http://www.globus.org

A.16.9 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

. . . . . .

A.17 nl_write(1)

A.17.1 NAME

nl_write - Write a NetLogger-formatted message.

A.17.2 SYNOPSIS

nl_write [options] name=value..

A.17.3 DESCRIPTION

Write one NetLogger-formatted message to standard output, TCP, or UDP. Any number of name=value pairs can be given as arguments. These will be copied to the output along with the standard values of ts=<timestamp> and event=<event_name>, to form a properly formatted log message.

A.17.4 OPTIONS

Note: single-letter options in upper-case control *how* things are logged, whereas lower-case options control *what* is logged.

--version show program's version number and exit

-h, --help show this help message and exit

-g, --guid add guid=GUID to message. This is overridden by an explicit guid=GUID argument.

-i, --ip add 'host=IP' to message. This is overridden by an explicit host=HOST argument.

-n NUM, --num=NUM Write NUM messages, each with n=<1..NUM> in them (default=1)

-H HOST, --host=HOST for UDP/TCP, the remote host (default=localhost)

-P PORT, --port=PORT For UDP/TCP, the port to write to (default=UDP 514, TCP 14380)

-S, --syslog add a header for syslog (default=False unless -U is given, then True)

-T, --tcp write message to TCP (default port=14380)

-U, --udp write message to UDP (default port=514)

A.17.5 EXAMPLES

To write the default message:

```
nl_write
```

To write a message with a host, guid, and attributes *foo* and *bar*:

```
nl_write -g -i foo=12345 bar='hello, world'"
```

To write a syslog-formatted message to the standard syslog UDP port (514):

```
nl_write -g -U msg='hello, world'
```

A.17.6 EXIT STATUS

Returns zero on success, non-zero on error

A.17.7 BUGS

The *host* option always uses the default interface.

There is no way to write a message with a user-defined timestamp, the time is always "now".

A.17.8 AUTHOR

Dan Gunter <dkgunter@lbl.gov>

B Index

R
R language, 29

S
SQL, 29